Seguridad en Arquitecturas de Procesadores

(Julio 2023)

Abramovich, Jolodovsky, Vainstein Aranguren.

Abstract—Este informe aborda la temática de seguridad en arquitecturas de procesadores. Se analizan casos de referencia importantes, como Spectre y Meltdown, que son amenazas que han planteado desafíos significativos en términos de protección y privacidad. También se exploran las áreas de investigación en curso y se proporcionan referencias avanzadas para continuar investigando sobre el tema. El objetivo de este informe es brindar a los profesionales de la computación una visión general de los desafíos y soluciones en la seguridad de las arquitecturas de procesadores.

- *Iara Jolodovsky - Estudiante de Lic. en Analisis de Sistemas - FIUBA - ijolodovsky@fi.uba.ar*
- *Martín Abramovich - Estudiante de Lic. en Analisis de Sistemas - FIUBA - mabramovich@fi.uba.ar*
- *Tomás Vainstein Aranguren - Estudiante de Lic. en Analisis de Sistemas - FIUBA - tvainstein@fi.uba.ar*

1 Introducción

En los últimos años, el estudio de las arquitecturas de procesadores ha ganado interés debido a la creciente preocupación por la seguridad en el diseño de sistemas informáticos. La evolución de las amenazas y ataques informáticos ha llevado a la identificación de vulnerabilidades importantes en los procesadores, lo que ha generado la necesidad de investigar y desarrollar soluciones efectivas.

2 Casos de referencia

2.1 Spectre

Spectre es una clase de vulnerabilidad de seguridad que explota las características de ejecución especulativa de los procesadores modernos. Estas vulnerabilidades permiten a los atacantes acceder a información confidencial almacenada en la memoria de otros procesos, lo que compromete la privacidad y la integridad de los sistemas informáticos. [1]

Hay dos variantes del ataque Spectre. La primera variante aprovecha las ramificaciones condicionales, donde el atacante entrena al predictor de ramificación para que prediga incorrectamente la dirección de una rama y ejecute especulativamente instrucciones que de otra manera no se habrían ejecutado. Esto permite al atacante leer información secreta almacenada en el espacio de direcciones de la víctima.

La segunda variante de los ataques Spectre explota las ramificaciones indirectas y se basa en el concepto de programación orientada a retornos (ROP). El atacante influye en la víctima para que ejecute especulativamente una secuencia de instrucciones desde el espacio de direcciones de la víctima. Los efectos de la ejecución especulativa en el estado nominal de la CPU se revierten eventualmente, pero los cambios en el estado de la memoria caché no se revierten, lo que permite que el gadget filtre información sensible a través de un canal secundario de caché.

Este tipo de amenazas aprovecha la forma en la que los procesadores modernos manejan la paginación de memoria para acceder a esa información. Cuando un proceso intenta acceder a una dirección virtual que no está presente en la memoria física, se produce una excepción de página. El sistema operativo maneja esta excepción y carga la página correspondiente en la memoria física. Los ataques Spectre pueden aprovechar esta excepción de página para acceder a información confidencial.

Estos ataques implican inducir a la víctima a realizar especulativamente operaciones que no ocurrirían durante la ejecución correcta del programa y que filtran información confidencial de la víctima a través de un canal lateral al adversario.

Esta capacidad de ejecución especulativa se encuentra en los microprocesadores de Intel, AMD y ARM que se utilizan en millones de dispositivos. Lo que se hace es aumentar el rendimiento adivinando probables rutas de ejecución futuras y prematuramente ejecutando las instrucciones en ellos. Por ejemplo, cuando el flujo de control del programa depende de un valor no almacenado en caché, ubicado en la memoria física, puede tomar varios ciclos de reloj antes de que se conozca el valor. En

lugar de desperdiciar estos ciclos, el procesador adivina la dirección del flujo de control, guarda un punto de control de su estado de registro y procede a ejecutar especulativamente el programa en la ruta adivinada.

Cuando el valor llega de la memoria, el procesador verifica su suposición. Si la hipótesis era incorrecta, el procesador descarta la ejecución especulativa al revertir el estado del registro del checkpoint almacenado. Sin embargo, en caso de que la hipótesis fuera correcta, los resultados de la ejecución especulativa se confirman produciendo una ganancia de rendimiento.
Desde una perspectiva de seguridad, la ejecución especulativa implica ejecutar un programa de formas posiblemente incorrectas. [1]

2.2 Meltdown

Meltdown es un novedoso ataque que permite superar el aislamiento de la memoria al proporcionar una forma sencilla para que cualquier proceso de usuario acceda al kernel del sistema operativo. Meltdown no explota ninguna vulnerabilidad de software, es decir, funciona en todos los principales sistemas operativos. En su lugar, utiliza la información de canal lateral disponible en la mayoría de los procesadores modernos, como las microarquitecturas de Intel desde 2010. [2]

Esta vulnerabilidad aprovecha esta característica para acceder a la memoria del kernel del sistema operativo, incluso si el atacante no tiene permisos de administrador en el sistema. Se puede utilizar para acceder a información confidencial aprovechando las vulnerabilidades en la ejecución fuera de orden[14] y la técnica de ataque de canal lateral basada en caché Flush+Reload [11], que aprovecha las diferencias de tiempo para inferir los datos accedidos desde una ubicación de memoria específica. A continuación, se describe cómo se lleva a cabo el ataque:

1. El atacante elige una ubicación de memoria que normalmente no sería accesible para él, como la memoria del kernel del sistema operativo.

2. El atacante carga el contenido de esa ubicación de memoria en un registro sin generar una excepción, a pesar de que la ubicación de memoria no es accesible. Esto se realiza aprovechando el pipeline y obteniendo de forma anticipada la dirección de memoria, lo cual no generaría una excepción.

3. A continuación, el atacante ejecuta una instrucción transitoria que accede a una línea de caché basada en el contenido secreto del registro. Esta instrucción transitoria no tiene un impacto real en el resultado del programa, pero afecta el estado microarquitectónico del procesador. Esta ejecución modifica el estado de la caché, marcando la línea accedida como válida e introduciendo posibles cambios o interferencias en otras partes del procesador que comparten la misma caché.

4. Después de ejecutar la instrucción transitoria, el atacante utiliza la técnica Flush+Reload para determinar qué línea de caché fue accedida. Esto se logra midiendo el tiempo de acceso a diferentes líneas de caché y comparando los tiempos.

Al repetir estos pasos para diferentes ubicaciones de memoria, el atacante puede determinar el contenido de la memoria del kernel del sistema operativo, incluyendo información confidencial como contraseñas, claves de cifrado u otros datos sensibles. [2]

2.2.1 Evaluación de Meltdown

Se ha evaluado el rendimiento de diversos sistemas ante un ataque Meltdown. [2]

En primer lugar se realizaron pruebas en diferentes sistemas, incluyendo sistemas con procesadores Intel, AMD y ARM, así como sistemas con diferentes versiones de Windows, Linux y Android.

También, se evaluó el impacto del ataque en el rendimiento del sistema utilizando diferentes cargas de trabajo, incluyendo pruebas de rendimiento de CPU y pruebas de rendimiento de Input/Output.

Asimismo, se estudió la capacidad del ataque para acceder a información confidencial, incluyendo contraseñas, claves de cifrado y otros datos sensibles.

De la misma forma, se analizó la efectividad de las contramedidas existentes, incluyendo parches de software y microcódigo, así como medidas de seguridad adicionales como el direccionamiento aleatorio de espacios de direcciones (ASLR) [8] y Kernel Address Space Layout Randomization (KASLR) [6].

Algunos resultados de estas evaluaciones mostraron que el impacto del ataque en el rendimiento del sistema puede ser significativo, especialmente en sistemas más antiguos.

En particular, se encontró que el ataque puede tener un impacto relevante en el rendimiento de la CPU y en el rendimiento de I/O.

También se demostró que las contramedidas existentes, incluyendo parches de software y microcódigo, pueden mitigar el riesgo de este tipo de ataques. En particular, se encontró que la implementación de la técnica Kernel Page Table Isolation (KPTI) [7] puede prevenir la filtración de información confidencial a través del ataque de Meltdown.

Por último, se pudo evidenciar que las medidas de seguridad adicionales, como ASLR y KASLR, pueden proporcionar una capa adicional de protección contra este tipo de ataques. Sin embargo, estas medidas no son suficientes para prevenir completamente el riesgo de los mismos. [2]

KPTI, ASLR y KASLR se detallan con mayor profundidad

en la sección de áreas de investigación en curso.

3 SOLUCIONES PROPUESTAS Y MEJORAS PRÁCTICAS

La seguridad de las arquitecturas de procesadores es un tema fundamental en la protección de la memoria del sistema y la prevención de vulnerabilidades y ataques maliciosos. Existen varias soluciones y mejores prácticas que se utilizan para fortalecer la seguridad en este ámbito, incluyendo el uso de técnicas como la memoria virtual y paginación, así como mecanismos de control de acceso como las listas de control de acceso (ACL).

A continuación se resumen los mecanismos de protección tradicionales para ilustrar con más contexto cómo los ataques de ejecución especulativa, Spectre y Meltdown, afectan la seguridad en los siguientes casos.

3.1 Memoria virtual

A pesar de que la principal función de la memoria virtual no es específicamente la seguridad, ésta puede contribuir a mejorar la defensa del sistema protegiendo y compartiendo recursos entre diferentes procesos en ejecución. Esta técnica permite asignar a cada proceso su propio espacio de direcciones virtuales, lo que crea una barrera entre ellos. Esto evita que alguno de estos procesos acceda o modifique directamente la memoria de otros procesos, lo que ayuda a prevenir interferencias y ataques de software malicioso entre diferentes programas en ejecución.

La protección se logra mediante el uso de la memoria virtual basada en páginas, donde se asignan páginas virtuales a páginas físicas en la memoria. Además, se utilizan mecanismos como TLB (Translation Lookaside Buffer) para agilizar los accesos a memoria. TLB es una memoria caché administrada por la unidad de gestión de memoria, que contiene partes de la tabla de paginación, la cual relaciona las direcciones lógicas con las físicas.

Por otro lado, los desbordamientos de búfer son vulnerabilidades comunes que pueden ser explotadas por atacantes para ejecutar código malicioso o manipular datos. En estos casos, la memoria virtual puede utilizar técnicas como ASLR para dificultar la explotación de estas vulnerabilidades, haciendo que sea más difícil para un atacante predecir la ubicación exacta de los datos o el código objetivo.

Además, la memoria virtual utiliza mecanismos de protección, como permisos de acceso (ACL), para controlar qué partes de la memoria pueden ser leídas, escritas o ejecutadas por cada proceso. Esto permite definir políticas de seguridad granulares y restringir el acceso a determinadas áreas de memoria, dificultando la obtención de información confidencial o la manipulación de datos críticos del sistema.

Por último, esta memoria también se utiliza para asignar y gestionar eficientemente los recursos del sistema como la memoria física, ayudando a prevenir situaciones en las que un proceso malicioso pueda agotar o interferir con los mismos.

Sin embargo, a pesar de estos mecanismos de protección, los sistemas operativos actuales contienen muchos errores y vulnerabilidades, lo que plantea desafíos adicionales en términos de seguridad. Por lo tanto, se han explorado modelos de protección más reducidos, como las máquinas virtuales, para mitigar estos riesgos y mejorar la seguridad del sistema.

Si bien la memoria virtual proporciona una capa de protección y compartición de recursos entre procesos, la seguridad en las arquitecturas de procesadores requiere una atención continua y el desarrollo de enfoques innovadores para garantizar la protección de la información y los sistemas.

En la Fig. 1. se muestra que el manejo de la memoria virtual permite la paginación de distintos procesos para que se asignen a una página física en común para compartir (1), las mismas páginas virtuales en diferentes procesos se asignen a diferentes páginas físicas (2) y algunas páginas virtuales residan en la memoria de intercambio en el disco (3). [3]

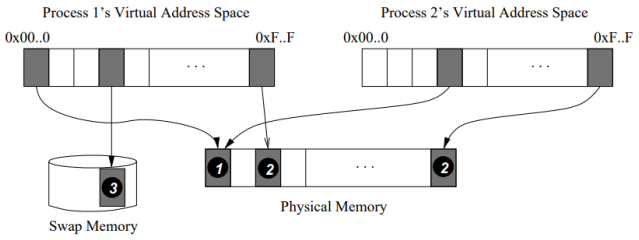

Fig. 1. Ejemplo de un esquema de paginación.3.2 Modo protegido

3.2 Modo protegido

El modo protegido es un modo de operación utilizado por los procesadores modernos en sistemas operativos multitarea y multiproceso. En este modo, se utilizan tablas de descriptores locales y globales para establecer límites de acceso a la memoria y privilegios de los programas.

Además, en el modo protegido se pueden establecer niveles de privilegio para los programas mediante el uso de anillos de protección. Por ejemplo, en Intel (AMD), se definen generalmente cuatro niveles de privilegio, siendo el nivel 0 el más alto (kernel) y el nivel 3 el más bajo (usuario). Los programas que se ejecutan en el nivel 0 tienen acceso completo a todos los recursos del sistema, mientras que los programas en los niveles superiores tienen acceso limitado. Sin embargo, es importante tener en cuenta que el número de niveles de privilegio puede variar según la arquitectura del procesador, e incluso dentro de la misma arquitectura Intel pueden existir únicamente dos niveles.

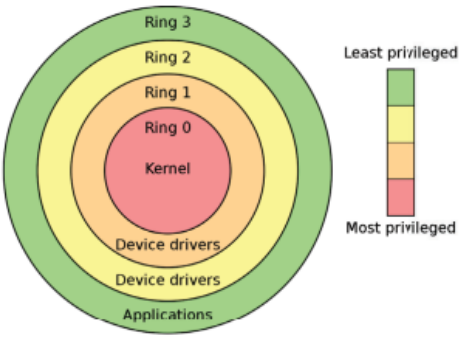

Fig 2. Diagrama sobre los distintos niveles de privilegio

En cuanto a la gestión de la memoria en este modo, los programas se ejecutan en un espacio de direcciones virtuales que se traducen a direcciones físicas mediante la Unidad de Gestión de Memoria (MMU, en inglés). La paginación es una técnica utilizada por la MMU para dividir la memoria virtual en bloques llamados páginas, las cuales se asignan a marcos de página física en la memoria RAM. La MMU utiliza una tabla de páginas para realizar la traducción de direcciones virtuales a direcciones físicas. [4]

3.3 Listas de control de acceso

Las ACL son un control de acceso discrecional que permiten asignar permisos granulares a diferentes objetos del sistema y controlar quién tiene acceso y qué operaciones pueden realizar. El concepto de matriz de acceso formulado por Butler W. Lampson [5] habla de una dimensión de la matriz que consiste en sujetos identificados que pueden intentar acceder a datos de los recursos. La otra dimensión lista los objetos que se pueden acceder. Esta matriz se puede descomponer por columnas dando listas de control de acceso. Para cada objeto una ACL enumera usuarios y sus derechos de acceso.

En el contexto de las vulnerabilidades Spectre y Meltdown, las ACLs por sí solas no son una solución directa. Estas vulnerabilidades son explotaciones de características de diseño en los microprocesadores y no se solucionan simplemente mediante el uso de ACLs.

Sin embargo, las ACLs pueden ser útiles en un enfoque de defensa en profundidad, que implica implementar múltiples capas de seguridad para mitigar riesgos. En este caso, las ACLs podrían usarse como parte de un conjunto de medidas de seguridad que buscan limitar el acceso a recursos y restringir la ejecución de código malicioso.

Por ejemplo, en el caso de Spectre y Meltdown, las ACLs podrían emplearse para controlar el acceso a ciertos archivos y directorios sensibles, como los relacionados con el kernel del sistema operativo. Limitar el acceso a estos recursos a usuarios privilegiados y a programas confiables podría ayudar a reducir el riesgo de explotación de estas vulnerabilidades.

4 ÁREAS DE INVESTIGACIÓN EN CURSO

4.1 Protección de canales laterales

Los canales laterales son formas indirectas de obtener información a través del análisis de datos, como el tiempo de acceso a la memoria o el consumo de energía. Las arquitecturas de procesadores deben abordar estos canales laterales para proteger la confidencialidad de los datos. En los casos de ataques de ejecución transitoria como Meltdown, la CPU continúa usando datos en la ejecución fuera de orden incluso si la carga de los datos resultó en una falla, por ejemplo, si falló la comprobación del nivel de privilegios.

Si bien los datos no se hacen visibles en el nivel de arquitectura, se codifican en el estado microarquitectónico, es decir, en la caché. A partir de ahí se hace visible en el plano arquitectónico usando ataques de canal lateral.

Entre las técnicas de seguridad que se han propuesto como soluciones para mitigar ciertos tipos de ataques existe KASLR. Ésta aleatoriza el diseño de la memoria permitiendo que sea más difícil predecir la ubicación de la memoria en el proceso, impidiendo el acceso del atacante a la misma. Específicamente en el caso del kernel, se aleatoriza su ubicación en el espacio de direcciones del sistema. [6]

A partir de diversos análisis sobre las mitigaciones de hardware implementadas por Intel para abordar específicamente Meltdown, se concluyó que estas no son completas y pueden ser explotadas para romper KASLR. La técnica llamada Echo Load [6] se utiliza para determinar si la dirección de memoria es válida o no. Lo que hace es calcular cuánto tarda una carga de memoria en completarse. Si la dirección de memoria es válida, la carga tarda menos tiempo que si no es válida. Una vez que se ha determinado que la dirección de memoria es válida, se puede utilizar para romper KASLR. La dirección de memoria se utiliza para calcular la ubicación de la memoria del kernel en el espacio de direcciones del sistema.

Por otra parte, KPTI (antes llamado KAISER) impone un aislamiento entre el espacio del kernel y el espacio del usuario para cerrar los canales laterales de hardware.
Se demostró que el sistema protege contra ataques de doble falta de página, ataques de canal lateral de prefetch y ataques de canal lateral basados en Transactional Synchronization Extensions (TSX) [12]. Además, KPTI tiene un impacto mínimo en el rendimiento del sistema, con un overhead de solo el 0,28% en promedio para todas las suites de pruebas.

Lo que se hace es asignar tablas de paginación separadas para el espacio del kernel y el espacio del usuario. [7]

Por último, existe ASLR que funciona como primera línea de defensa, seleccionando ubicaciones aleatorias en la memoria virtual de un proceso protegido para colocar código o datos. Esta técnica de protección a diferencia de KASLR se aplica a los procesos de usuario. [8]

4.2 Aislamiento y Sandboxing

Mejorar el aislamiento entre diferentes procesos y aplicaciones es esencial para evitar ataques de escalada de privilegios. El desarrollo de técnicas de sandboxing efectivas puede ayudar a limitar el impacto de las vulnerabilidades y proteger los sistemas de posibles ataques.

El sandboxing [13] es una técnica que se utiliza para aislar aplicaciones y procesos en entornos controlados, limitando su acceso a recursos y datos sensibles. Los investigadores están trabajando en el desarrollo de sandboxing mejorado, que incluye técnicas como la virtualización ligera y los contenedores seguros. Estos enfoques permiten un mayor grado de aislamiento y control sobre las aplicaciones, evitando que un proceso comprometido afecte a otros procesos o al sistema en general.

Dentro del área de los programas que utilizan la tecnología de contenedores seguros se encuentran Docker, Docker Swarm, Kubernetes, Nomad y Podman. Este último, a diferencia de los demás, no depende de un daemon, sino que inicia contenedores y pods como procesos secundarios. [15]

Uno de los programas más utilizados en este ámbito es Docker [9], el cual se ejecuta en el kernel del sistema operativo sin utilizar virtualización. Su función es la creación de estos contenedores o entornos separados del sistema operativo para poder ejecutar aplicaciones y realizar procesos en entornos controlados, lo cual trae beneficios como velocidad, portabilidad, escalabilidad, entrega rápida y densidad.

Al ejecutar las aplicaciones de forma aislada, Docker le otorga a cada contenedor su propio entorno y recursos separados. Esto ayuda a prevenir la interferencia entre contenedores y limita el impacto de posibles vulnerabilidades en una aplicación en particular.

Si bien Docker es visto como superior a otras tecnologías de contenedores y la creación de máquinas virtuales para ejecutar programas debido a su menor sobrecarga de recursos, también tiene desventajas como el hecho de que solo puede ejecutarse en máquinas de 64 bits.

Una herramienta útil en el área de aislamiento de procesos es SonarQube [10], la cual se ejecuta en contenedores seguros como Docker y sirve para inspeccionar el código en un entorno de contenedores sin utilizar una máquina virtual separada y así identificar errores, hediondez de código y vulnerabilidades de seguridad comunes, como inyección de código, acceso inseguro a datos o uso incorrecto de funciones criptográficas. Al proporcionar una visibilidad detallada de los problemas de seguridad en el código, SonarQube ayuda a los desarrolladores a identificar y solucionar estas vulnerabilidades antes de que se desplieguen en un entorno de producción.

Es importante destacar que tanto Docker como SonarQube son herramientas complementarias que pueden utilizarse en conjunto para mejorar la seguridad en el desarrollo y despliegue de aplicaciones. Docker ofrece características de aislamiento y control de acceso para proteger los recursos del sistema en los contenedores, mientras que SonarQube se enfoca en la identificación y mitigación de vulnerabilidades en el código fuente. Al utilizar estas herramientas en combinación con buenas prácticas de seguridad, los desarrolladores pueden fortalecer la seguridad de sus aplicaciones y mitigar posibles riesgos de vulnerabilidades.

5 Conclusión

En este informe se ha examinado la temática de seguridad en arquitecturas de procesadores, centrándose en los casos de referencia de Spectre y Meltdown, que representan desafíos significativos en términos de protección y privacidad. Estos ataques han demostrado la importancia de abordar las vulnerabilidades en los procesadores modernos y han generado la necesidad de investigar y desarrollar soluciones efectivas.

Estas vulnerabilidades han demostrado que incluso los sistemas operativos y las contramedidas existentes no son suficientes para garantizar la seguridad de las arquitecturas de procesadores.

Las evaluaciones indagadas en este informe han revelado el impacto significativo que estos ataques pueden tener en el rendimiento del sistema, especialmente en sistemas más antiguos. Sin embargo, se han propuesto soluciones y mejores prácticas para fortalecer la seguridad en las arquitecturas de procesadores.

No obstante, es importante destacar que la seguridad de las arquitecturas de procesadores sigue siendo un desafío en constante evolución. Se requiere una investigación continua y un enfoque proactivo para abordar las vulnerabilidades emergentes y garantizar la protección de la información confidencial.

6 Referencias

[1] Kocher, P., Genkin, D., Gruss, D., Haas, W., Hamburg, M., Lipp, M., ... & Yarom, Y. (2019). Spectre attacks: Exploiting speculative execution. Communications of the ACM, 62(11), 96-107.
[2] Lipp, M., Schwarz, M., Gruss, D., Prescher, T., Haas, W., Fogh, A., ... & Werner, M. (2018). Meltdown: Reading kernel memory from user space. In 27th USENIX Security Symposium (USENIX Security 18) (pp. 973-990).
[3] "Computer Architecture: A Quantitative Approach" de John L. Hennessy y David A. Patterson.
[4] Rudimento de un sistema operativo para la arquitectura x86 64: parte en modo protegido - Jimenez, Universidad Politécnica de Madrid
[5] "Computer Security: Principles and Practice" by William Stallings and Lawrie Brown.
[6] KASLR: Break It, Fix It, Repeat - Daniel Gruss.
[7] KASLR is Dead: Long Live KASLR - Daniel Gruss.
[8] ASLR on the Line: Practical Cache Attacks on the MMU - Ben Grass.
[9] An Introduction to Docker and Analysis of its Performance - Babak Bashari Rad, Harrison John Bhatti, Mohammad Ahmadi
[10] An Introduction To SonarQube: How to setup SonarQube for static code analysis of your codebase - Prince Mittal

[11] FLUSH+RELOAD: a High Resolution, Low Noise, L3 Cache Side-Channel Attack, Yuval Yarom Katrina Falkner. The University of Adelaide.
[12] Performance Evaluation of Intel R Transactional Synchronization Extensions for High-Performance Computing, Richard M. Yoo, Christopher J. Hughes, Konrad Lai, Ravi Rajwar.
[13] Sandboxing Applications, Vassilis Prevelakis, Diomidis Spinellis.
[14] Impacto de las vulnerabilidades Meltdown y Spectre en las grandes empresas: M. Paula Baldivia, Franco D. Fresno, Matías Heidenreich, Juan Pablo Pinto, Julieta R. Prado Walsh.
[15] Podman vs Docker: What are the differences? - Alex Gamela, Rute Figueiredo