La Inseguridad del Hardware Actual: Un Análisis de Vulnerabilidades y Mitigaciones

Alfonso Pili, Iñaki Vydra, Santiago Barrionuevo

Resumen—La seguridad del hardware actual se ha convertido en una preocupación crítica en la era digital. Este informe profundiza en las vulnerabilidades que afectan a los procesadores contemporáneos, incluyendo ataques de canal lateral como Spectre y Meltdown, que explotan las optimizaciones de rendimiento para acceder a información sensible. Se analizan las técnicas de mitigación implementadas, como el aislamiento de tablas de páginas (KPTI), la aleatorización del diseño del espacio de direcciones del kernel (KASLR) y Retpoline, que buscan contrarrestar estos ataques.

Index Terms—Seguridad del hardware, Vulnerabilidades del procesador, Ataques de canal lateral, Spectre, Meltdown, Predicción de saltos, Ejecución especulativa, Mitigaciones de hardware, Mitigaciones de software, Prevención de ataques, KPTI, KASLR, Retpoline.

1. Introducción

La seguridad de la información es un tema cada vez más relevante en la era digital. Los procesadores son el corazón de cualquier sistema informático, y su seguridad es crucial para garantizar la integridad y la confidencialidad de los datos. Sin embargo, la complejidad de la microarquitectura de los procesadores modernos y la constante evolución de las amenazas informáticas hacen que la seguridad de estos dispositivos sea un desafío cada vez más complejo.

2. Optimizaciones

El rendimiento de los procesadores modernos ha mejorado gracias a varias técnicas. Como ya no es tan fácil seguir reduciendo el tamaño de los transistores o aumentar la frecuencia del reloj, se han buscado otras soluciones a nivel de arquitectura. La clave está en el paralelismo: optimizar y extender las rutas de instrucciones para ejecutar más operaciones al mismo tiempo dentro de un hilo, además de aumentar la cantidad de núcleos físicos y lógicos en un procesador [5].

Las optimizaciones clave de los procesadores modernos son: la **ejecución especulativa**, **ejecución fuera de orden** y la **predicción de bifurcaciones** (Branch Prediction) [1].

- **Ejecución especulativa**: Permite al procesador ejecutar instrucciones ”por adelantado”**(aprovechando el paralelismo y la predicción de bifurcaciones)** para evitar tiempos de espera. Si la predicción es incorrecta, los resultados especulativos se descartan, aunque pueden dejar rastros en la caché, lo que ha dado lugar a vulnerabilidades.
- **Ejecución fuera de orden**: Aprovecha al máximo los recursos del procesador ejecutando instrucciones en paralelo en distintas unidades de ejecución. Mientras una unidad está ocupada, las demás pueden procesar otras instrucciones de forma simultánea, siempre que no se rompan las reglas arquitectónicas impuestas por el hardware.
- **Predicción de bifurcaciones**: Es una técnica utilizada para decidir qué rama de un condicional es más probable que se ejecute. Esto optimiza el flujo de instrucciones al anticiparse al resultado de la condición, evitando interrupciones en el pipeline del procesador y mejorando el rendimiento general.

Los procesadores modernos implementan estas técnicas de optimización para mejorar el rendimiento. Sin embargo, estas mismas técnicas pueden ser explotadas para crear **canales laterales**, de los cuales hablaremos más adelante en la sección de **Vulnerabilidades**.

3. Niveles de Privilegios

Antes de hablar de las vulnerabilidades por explotación de canales laterales de caché, aprovechando las optimizaciones del microprocesador, es fundamental entender cómo se gestionan los privilegios de las instrucciones para interactuar con la CPU y la memoria, ya que esto influye directamente en la seguridad del sistema y en la posibilidad de ataques que exploten estas interacciones.
En la Fig. 1 se pueden apreciar 4 anillos de seguridad en las arquitecturas x86, aunque normalmente los sistemas operativos solo utilizan el nivel 0 y el nivel 3, lo que les permite utilizar únicamente 1 bit para verificar el modo en el cual debe trabajar el procesador.
Cada modo tiene una región de memoria mapeada diferente; ese aislamiento está dado por la **virtualización de la memoria** [6]. Por lo tanto, el modo usuario no puede acceder a la misma información que el modo kernel, al menos de manera directa.

El **modo usuario** corresponde al anillo 3, el cual solo ejecuta código a nivel software. Es decir, está aislado tanto del sistema operativo como del kernel; sus recursos son limitados y, por cada intento de modificación de memoria, manejo de archivos o contacto con el hardware, se realiza

Alfonso Pili - - FIUBA - (mail: .uba.ar)
Iñaki Vydra - Ingenieria Informatica - FIUBA - (mail: ivydra@fi.uba.ar)
Santiago Barrionuevo - - FIUBA - (mail: .uba.ar)

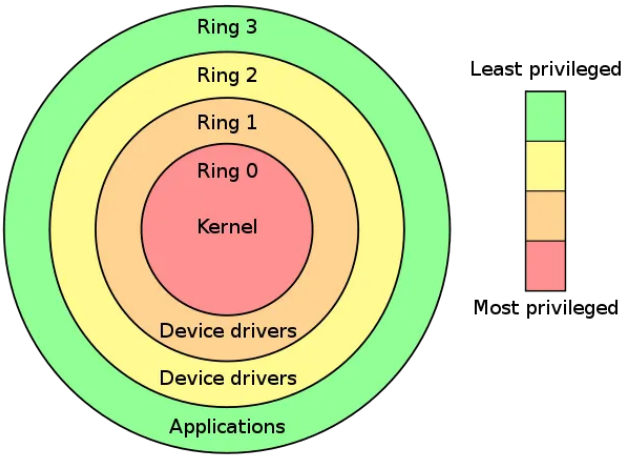

Figura 1. Distribución jerárquica de los privilegios.

una **syscall**. Estas llamadas al sistema permiten al programa cambiar al modo kernel, dejando que el sistema operativo ejecute las instrucciones necesarias y luego vuelva al modo usuario sin violar la seguridad.
De aquí podemos deducir que, en el anillo de nivel 0, se ejecuta únicamente código a nivel kernel. Este nivel es administrado por el **modo kernel** y es el nivel de seguridad más importante. El modo kernel tiene contacto directo con el hardware, es decir, tiene acceso a todo el mapeado de la memoria, incluidas las tablas de páginas, donde se encuentra nuestra información más sensible, como contraseñas, credenciales y demás datos privados.
Por lo tanto, el principal problema aquí es que, si algún programa malicioso lograra ejecutar instrucciones en modo kernel o explotara la ejecución especulativa y fuera de orden para ejecutar instrucciones sin los privilegios necesarios, se pondría en riesgo toda nuestra seguridad, ya que podrían acceder a las páginas del kernel sin siquiera tener los privilegios adecuados.

4. VULNERABILIDADES

La seguridad de los sistemas puede verse comprometida por distintos tipos de vulnerabilidades. En este artículo, haremos hincapié en los **ataques de canales laterales** [11] (del inglés Side Channel Attacks), que explotan información derivada de las condiciones físicas o lógicas durante la ejecución de un sistema para extraer datos sensibles.

¿A qué nos referimos con çondiciones físicas y lógicas¿ A que estos ataques no se basan en vulnerabilidades directas en algoritmos o software, sino en la medición de:

Condiciones físicas: Fenómenos observables en el hardware, como el consumo de energía, el tiempo de ejecución de operaciones o las emisiones electromagnéticas/acústicas.

Condiciones lógicas: Los comportamientos de la implementación del sistema, como el estado de la memoria caché, el mapeo de memoria o los patrones de acceso a recursos compartidos.

Estas fugas de información indirecta son los llamados **canales laterales**, que permiten al atacante obtener datos secretos sin acceder directamente a ellos.
Existen muchísimos casos de canales laterales que se pueden explotar [12], pero en este informe solo vamos a entrar en el canal de temporización de caché (cache timing side-channel), el cual es utilizado por Meltdown y Spectre.

4.1. Spectre

Spectre es una vulnerabilidad que explota la ejecución especulativa para engañar al procesador y hacer que acceda a datos sensibles en la memoria, rompiendo el aislamiento entre procesos. [4]
Afecta casi todos los procesadores modernos (Intel, AMD, ARM).
Fue descubierto por dos equipos de investigadores: Jann Horn (Google Project Zero) y Paul Kocher, en colaboración con Daniel Genkin (University of Pennsylvania, University of Maryland), Mike Hamburg (Rambus), Moritz Lipp (Graz University of Technology) y Yuval Yarom (University of Adelaide, Data61).
Funciona de la siguiente manera: Primero, el atacante manipula la unidad de predicción de ramas del procesador para inducir una predicción incorrecta sobre las instrucciones que se ejecutarán. De esta manera, el procesador se prepara para ejecutar instrucciones especulativamente basadas en esa suposición errónea. A continuación, el procesador, confiando en su predicción equivocada, ejecuta especulativamente instrucciones que no debería ejecutar. Estas instrucciones pueden incluir accesos a áreas de memoria privadas, como datos de otros procesos o del sistema operativo. Aunque el procesador eventualmente detecta el error y descarta estas instrucciones, los efectos de esos accesos permanecen en la memoria caché. Por último, el atacante explota estos efectos secundarios midiendo los tiempos de acceso a la caché con técnicas como Flush+Reload. Si el acceso a una dirección de memoria es más rápido, significa que los datos fueron cargados especulativamente y están en la caché. Si el acceso es más lento, significa que los datos no estaban en caché. A través de estos tiempos de acceso, el atacante puede reconstruir información sensible sin necesidad de acceder directamente a la memoria protegida [4].
A continuación, presentaremos 2 variantes de Spectre:

4.1.1. *Spectre-PHT*

En este tipo de ataque, el atacante manipula el predictor de saltos para que el procesador especule incorrectamente sobre una comparación en un programa. En el siguiente código de C:

```
if (x < array1_size)
    y = array2[array1[x] * 4096];
```

Figura 2. Fragmento de código en C ilustrando el concepto de manipulación de predicción de saltos.

x es un valor controlado por el atacante. Normalmente, el procesador verifica que x esté dentro de los límites de array1, pero si el predictor de saltos ha sido entrenado para asumir que x es siempre válido, el procesador accederá a array1[x] sin verificar los límites. Si x es un valor malicioso, puede hacer que array1[x] apunte a una dirección confidencial. Luego, array2[array1[x] * 4096] almacenará

esa información confidencial en la caché, permitiendo que el atacante lo extraiga mediante mediciones de tiempo de acceso [4].

4.1.2. *Spectre-BTB*

En esta variante, el atacante manipula la Branch Target Buffer (BTB), que es la estructura del procesador encargada de predecir el destino de saltos indirectos. El atacante entrena el BTB para que un salto indirecto en el código del programa víctima se redirija erróneamente a una ubicación de código que el atacante elige. Esta ubicación puede contener una secuencia de instrucciones (un "gadget") que filtra información confidencial a través de la caché.

4.2. Meltdown

Meltdown es una vulnerabilidad que permite a un atacante acceder a la memoria protegida del sistema, incluyendo la memoria del kernel, rompiendo los mecanismos de aislamiento entre aplicaciones y el sistema operativo [2].
Fue descubierto por tres equipos diferentes: Jann Horn (Google Project Zero), Werner Haas y Thomas Prescher (Cyberus Technology) y Daniel Gruss, Moritz Lipp, Stefan Mangard y Michael Schwarz (Graz University of Technology).
Esta vulnerabilidad afecta principalmente a los procesadores Intel fabricados desde 1995, aunque ciertas variantes también pueden impactar a algunos modelos de ARM y AMD.
Funciona de la siguiente manera:
Primero, el atacante manipula el programa para que el procesador cargue en un registro el contenido de una dirección de memoria secreta, a la que normalmente no tendría acceso directo. Luego, debido a la ejecución especulativa, el procesador accede anticipadamente a la memoria caché basándose en ese contenido secreto, dejando un rastro en la caché. Finalmente, el atacante explota este rastro utilizando la técnica Flush+Reload, que le permite monitorear la caché y detectar qué datos han sido accedidos recientemente, revelando así información confidencial [2].
A continuación, presentaremos 3 variantes de Meltdown:

4.2.1. *Meltdown-P*

Este ataque afecta a los procesadores Intel y explota una vulnerabilidad en el sistema SGX, que es una tecnología que crea espacios seguros en la memoria donde se pueden almacenar datos confidenciales.
El ataque se basa en manipular la tabla de páginas, que es un mapa que le indica al procesador qué dirección tiene cada pieza de información. Cuando un atacante cambia alguna de estas direcciones en la tabla, se provoca un fallo de protección. Este fallo debería impedir el acceso a los datos y bloquear el sistema, pero debido a la ejecución especulativa, el procesador sigue trabajando aunque haya ocurrido un error. A pesar de detectar el fallo, el procesador continúa ejecutando instrucciones usando direcciones de memoria incorrectas, lo que hace que los datos se carguen temporalmente en la caché L1. Luego, el atacante puede usar la técnica Flush+Reload para extraer esta información.

4.2.2. *Meltdown-NM*

Este ataque aprovecha cómo la CPU maneja de manera ineficiente los registros de la unidad de punto flotante (FPU) al cambiar de un proceso a otro. En lugar de guardar los registros de inmediato cuando se cambia de proceso, la CPU los marca como "no disponiblesz los guarda solo cuando se vuelven a utilizar.
El ataque se desarrolla en tres pasos:
1- La víctima carga datos en la FPU.
2- La CPU cambia al atacante y marca la FPU como "no disponible".
3- El atacante ejecuta una instrucción que usa la FPU, lo que genera una excepción. Sin embargo, antes de que la CPU detenga la ejecución, ya ha procesado temporalmente instrucciones usando datos de la víctima.
De esta manera, el atacante puede aprovechar los datos no guardados y extraer información sensible que se encontraba en los registros de la FPU, sin tener acceso directo a ella [3].

4.2.3. *Meltdown-PK*

Los procesadores Intel Skylake-SP para servidores incluyen una función de protección de claves de memoria (PKU), que permite modificar los permisos de acceso a páginas de memoria dentro de un proceso sin necesidad de realizar una llamada al sistema o al hipervisor. Esto facilita un aislamiento eficiente de datos sensibles.
Meltdown-PK es un ataque que evade las restricciones de lectura y escritura impuestas por PKU. Aunque un atacante no tenga permisos para modificar los permisos de acceso con la instrucción wrpkru, puede explotar la ejecución especulativa del procesador para acceder temporalmente a datos protegidos y filtrarlos a través de un canal encubierto [3].

5. MITIGACIONES

Las vulnerabilidades de Spectre y Meltdown surgen como consecuencia directa de optimizaciones en los procesadores modernos. Aunque estas técnicas se enfocan en mejorar el rendimiento, también han introducido riesgos de seguridad que han permitido el desarrollo de ataques capaces de comprometer la seguridad de los sistemas.

Dado que el problema está relacionado con el diseño de la arquitectura de los procesadores, no tendremos una solución definitiva hasta que el hardware sea modificado. Sin embargo, debido al tiempo que toma el desarrollo y fabricación de nuevos chips, podrían pasar varios años antes de que las futuras generaciones de CPUs solucionen completamente el problema [4] [2].

Mientras tanto, se han desarrollado diversas mitigaciones a nivel de software y hardware que pueden controlar algunas variantes de estas vulnerabilidades y minimizar el riesgo de ataques en sistemas actuales. En este informe vamos a listar algunas de las más relevantes.

5.1. Aislamiento de Tablas de Páginas y Aleatoridad

El kernel se encarga de establecer tablas de páginas para controlar la asignación entre una dirección virtual y una dirección física. El acceso entre el núcleo y el espacio de usuario también se controla mediante las tablas de páginas. Una página que está mapeada para el kernel no es accesible

desde el espacio de usuario, aunque el núcleo normalmente puede acceder al espacio de usuario.

Traducir los mapeos puede ser costoso, por lo que el hardware tiene un búfer de traducción de direcciones (TLB) para almacenar asignaciones. A veces es necesario eliminar las asignaciones antiguas (invalidar el TLB), pero hacerlo es costoso, por lo que el código se escribe para minimizar tales llamadas. Un truco aquí es mantener siempre las tablas de páginas del núcleo mapeadas cuando los procesos de usuario se están ejecutando. Los permisos de la tabla de páginas impiden que el espacio de usuario acceda a las asignaciones del kernel, pero el núcleo puede acceder a ellas inmediatamente cuando se realiza una llamada al sistema.

Exploits como Meltdown demostraron que este método de mantener las tablas de páginas del kernel disponibles en el espacio de usuario es vulnerable. Si bien no permite directamente la filtración de información del núcleo, puede ser explotado por atacantes. Estos generan intentos de lectura sobre dichas páginas; al no contar con los permisos necesarios, la CPU bloquea el programa por seguridad (ya que se está ejecutando en modo usuario). Sin embargo, la CPU prebusca los datos de todas formas y no los desecha, por lo que quedan en la caché. Este comportamiento es aprovechado por los atacantes para medir el tiempo de acceso a los datos prebuscados y, de esta manera, llevar a cabo un ataque de canal lateral basado en caché.

5.1.1. *Mitigación Meltdown*

Para mitigar vulnerabilidades explotadas por ataques de canal lateral como Meltdown, se implementó **Kernel Page Table Isolation**[9], [8]. Fue un cambio total en la gestión de memoria. En lugar de compartir una única tabla de páginas con el kernel, ahora cada aplicación tiene dos conjuntos de tablas de páginas (fig. 4), y el sistema cambia entre ellas en cada transición entre el modo usuario y modo kernel. Cuando el código en modo usuario intenta acceder a una dirección del núcleo, no va a encontrar una dirección válida en la tabla de páginas, evitando así la explotación de Meltdown (Fig. 3). Pero este cambio tiene un gran impacto en el rendimiento. Por cada cambio de contexto (syscall, interrupciones y excepciones), se deben intercambiar las tablas de páginas, lo que implica limpiar el **búfer de traducción de direcciones** (TLB). Cuando este es vaciado, el procesador debe recorrer las tablas de páginas en memoria para encontrar la dirección física correspondiente, lo que puede ser entre 10 y 100 veces más lento en caso de fallos de TLB. Este impacto se nota mucho en aplicaciones con llamadas al sistema frecuentes, como las que manejan big data, redes o almacenamiento intensivo. En los procesadores más actuales, este problema se mitiga mediante **Process Context Identifiers**[14] (PCID). Esta funcionalidad asocia cada entrada del TLB con un conjunto de tablas de páginas específico. Los intercambios de tablas de páginas no eliminan completamente el TLB, por lo tanto, reducen la penalización de rendimiento de KPTI sin romper la seguridad.

5.1.2. *KASLR*

Otra estrategia de defensa es **Kernel Address Space Layout Randomization**[13] (KASLR), que se basa en aleatorizar la dirección donde se carga el kernel en memoria,

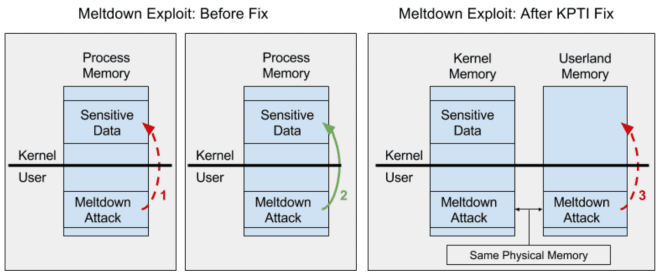

Figura 3. Ilustración de cómo Meltdown intenta vulnerar las tablas de páginas antes y después de utilizar KPTI.

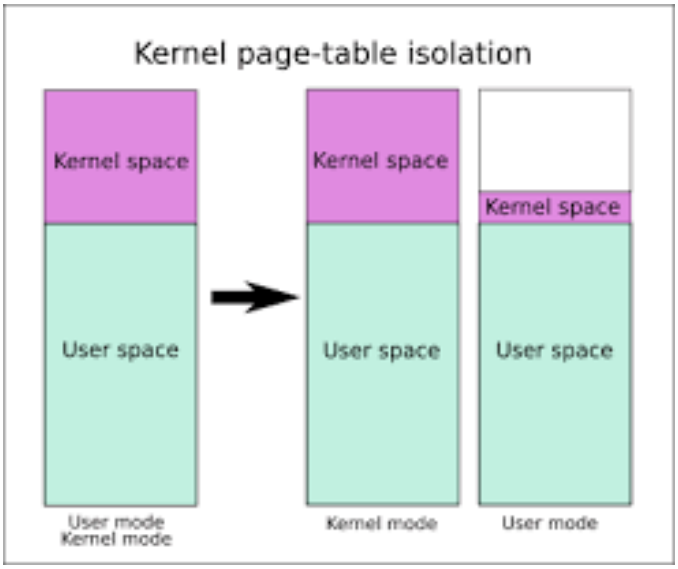

Figura 4. Imagen representativa de cómo se modifican las tablas del modo usuario y kernel, luego de aplicarle el aislamiento. Fíjese que en las tablas del modo usuario aún quedan algunas páginas del modo kernel. Esto se debe a que el sistema operativo necesita acceso a ciertas páginas del núcleo para el correcto funcionamiento del programa y utilizar mecanismos como las syscall para las interacciones entre los dos modos.

haciendo más difícil que los atacantes predigan su ubicación y hagan ataques basados en ejecución especulativa o acceso a caché. Esta estrategia aleatoriza las direcciones en cada arranque del sistema, eligiendo dinámicamente una ubicación para el núcleo. Para la implementación, se han desarrollado técnicas como los parches de Cook, que modifican el proceso de arranque para determinar la dirección más baja segura en la que puede colocarse el kernel. Luego, el sistema recorre las regiones de memoria e820 y cuenta los espacios del tamaño adecuado para el kernel. A partir de estos espacios, se elige uno al azar utilizando la mejor fuente de números aleatorios disponible, ya sea la instrucción RDRAND, los bits menos significativos del contador de tiempo RDTSC o bits de los puertos de E/S del temporizador. Una vez seleccionado el espacio, el kernel se descomprime, se maneja la reubicación y se inicia el sistema.

De esta manera, no solo se aísla el acceso a las páginas del kernel con KPTI y KASLR, sino que también se asegura que la ubicación del kernel en memoria sea impredecible en cada arranque.

Sin embargo, aunque estas técnicas reducen significativamente la efectividad de ataques como Meltdown y Spectre, no son soluciones definitivas. Ataques de canal lateral más sofisticados pueden seguir filtrando información del kernel a pesar de estas mitigaciones, lo que ha llevado a la investigación de nuevas estrategias de defensa, tanto en software como en hardware.

5.2. Retpoline

El nombre Retpoline es un acrónimo de "return" (retorno) y "trampoline" (trampolín). En este contexto, un trampolín es un pequeño fragmento de código invocado como función, que realiza una tarea breve y, a diferencia de una llamada a función común, no regresa a su llamador después de la ejecución, sino que salta a otra ubicación, como se muestra en la Fig. 5. Si el destino del salto del trampolín es una función, al retornar, dicha función regresará al llamador original del trampolín.[15][16]

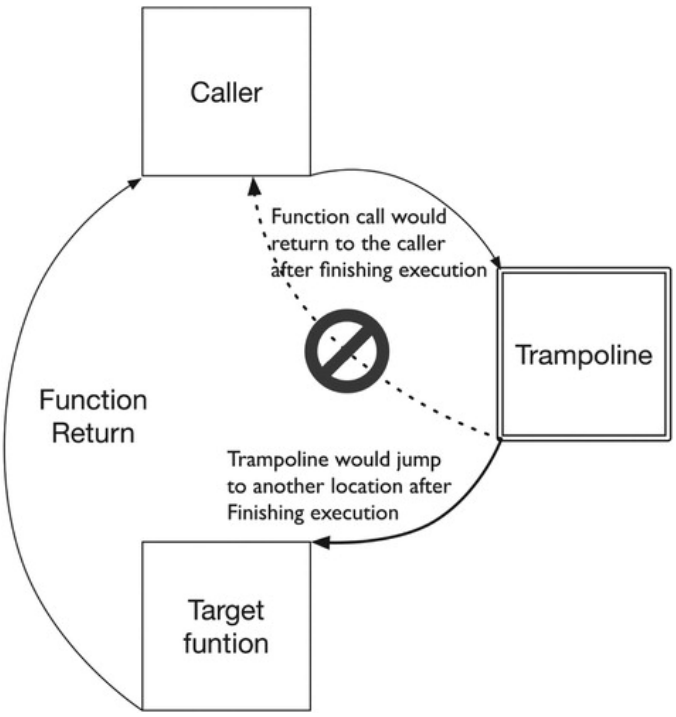

Figura 5. Trampoline se refiere a un fragmento de código que saltaría a otra ubicación después de terminar la ejecución en lugar de regresar al llamador original.

Retpoline fue desarrollado por Google para mitigar los ataques Spectre-BTB. Su objetivo es evitar que este tipo de ataques manipulen el Branch Target Buffer (BTB) y desvíen la ejecución especulativa hacia una dirección arbitraria. Para ello, Retpoline reemplaza las bifurcaciones indirectas con una secuencia especial de instrucciones basada en llamadas y retornos, asegurando que la ejecución especulativa permanezca en un bucle inofensivo dentro del Return Stack Buffer (RSB). Esto impide que un atacante manipule las predicciones de bifurcaciones para ejecutar código malicioso.

Para lograr esto, Retpoline aprovecha la forma en que los procesadores gestionan las llamadas y retornos de funciones. Al utilizar una instrucción call hacia un trampolín, se llena el RSB con una dirección de retorno controlada. Cuando el procesador especula sobre el retorno, en lugar de seguir una predicción potencialmente manipulada desde el BTB, sigue la dirección almacenada en el RSB y ejecuta un bucle de instrucciones seguras.[15][16]

REFERENCIAS

[1] Hennessy, J. L., Patterson, D. A. (2006). Computer Architecture: A Quantitative Approach (4a ed.). Morgan Kaufmann.
[2] Lipp, M., Schwarz, M., Gruss, D., Prescher, T., Haas, W., Fogh, A., Horn, J., Mangard, S., Kocher, P., Genkin, D., Yarom, Y. y Hamburg, M. (2018). Meltdown: Reading Kernel Memory from User Space. In *27th USENIX Security Symposium (USENIX Security 18)*.
[3] Claudio Canella , Jo Van Bulck, Michael Schwarz, Moritz Lipp, Benjamin von Berg, Philipp Ortner, Frank Piessens, Dmitry Evtyushkin y Daniel Gruss. *A Systematic Evaluation of Transient Execution Attacks and Defenses*. 2019. [Disponible]: https://www.usenix.org/conference/usenixsecurity19/presentation/canella
[4] Paul Kocher, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, y Yuval Yarom. *Spectre Attacks: Exploiting Speculative Execution*. 2019. [Disponible]: https://spectreattack.com
[5] Paralelismo en las instrucciones, [Disponible]: https://orgacomp.github.io/9557/teoria/procesador/paralelismo/
[6] Cap 9, Computer Systems: A Programmer's Perspective (3.ª ed.) 2015 , Pearson.
[7] Linux system calls," 2019. [Disponible]: http://man7.org/linux/man-pages/man2/syscalls.2.html
[8] Aleatorización del diseño del espacio de direcciones del núcleo, [Disponible]: https://bit.ly/aleatorizacion-kaslr
[9] Page Table Isolation, The Linux Kernel, [Disponible]: https://bit.ly/PageTableIso.
[10] Performance Impact of Host Kernel Page Table Isolation on Virtualized Servers, 2021, Song Wei; Kun Zhang; Bibo Tu, [Disponible]: https://ieeexplore.ieee.org/document/9421326
[11] Introduction to Side-Channel Attacks; Paul Kocher, Joshua Jaffe, Benjamin Jun.
[12] Power Analysis Attacks: Revealing the Secrets of Smart Cards, 2007, Stefan Mangard, Elisabeth Oswald, Thomas Popp.
[13] KASLR in the age of MicroVMs, 2022, [Disponible]: https://www.cc0x1f.net/publications/kaslr.pdf
[14] Process Context IDentifiers, [Disponible]: https://www.kernel.org/doc/Documentation/x86/pti.txt, https://bit.ly/pciddoc
[15] Chen, B., Wu, Q., Tan, Y., Yang, L., Zou, P. (2018). Exploration for software mitigation to spectre attacks of poisoning indirect branches. IETE Technical Review, 35(sup1), 119–127. https://doi.org/10.1080/02564602.2018.1531072
[16] M. F. Abdul Kadir, J. K. Wong, F. Ab Wahab, A. F. A. Abidin Bharun, M. A. Mohamed and A. H. Zakaria, Retpoline Technique for Mitigating Spectre Attack,"2019 6th International Conference on Electrical and Electronics Engineering (ICEEE), Istanbul, Turkey, 2019, pp. 96-101, doi: 10.1109/ICEEE2019.2019.00026.