

Tema 17: Conceptos de Seguridad

Arquitectura de Computadoras

Ing. Nicolás Majorel Padilla (npadilla@herrera.unt.edu.ar)

<http://microprocesadores.unt.edu.ar/arqcom/>

Temas que veremos

- ▶ Estado “actual” en Arquitectura de Computadoras y el problema “nuevo”.
- ▶ Ataques de tipo *side-channel*.
- ▶ Ataques de Ejecución Transitoria.
 - Meltdown, Spectre y otros.
- ▶ Ataques tipo Rowhammer.
- ▶ Paliativos y sus limitaciones.
- ▶ Zonas de Ejecución Segura (o no tanto?).
- ▶ Corrupción de Datos Silenciosa.
- ▶ Perspectivas a futuro.

¿Dónde estamos parados?

- ▶ La industria fue impulsada durante décadas por la búsqueda continua de mayor performance.
- ▶ **ILP**: Pipelining, Ejecución fuera de Orden, Ejecución especulativa, Predicción de saltos.
- ▶ Jerarquía de cachés.
- ▶ Multithreading, Multiprocesamiento.
- ▶ Procesadores actuales están compuestos por varios núcleos.
- ▶ Usualmente con memoria compartida, con protocolos para mantener coherencia.

Repaso: ISA

- ▶ Un ISA establece el “contrato” entre el software y el hardware por debajo.
 - ▶ Define todo el repertorio de instrucciones.
 - ▶ Define niveles de ejecución privilegiadas (Usuario/Supervisor).
 - ▶ Define mecanismos utilizados por el SO (Manejo de Excepciones, cambios de proceso, etc).
 - ▶ Define el modelo de memoria.
- ▶ Un ISA tradicionalmente no contempla optimizaciones al hardware.
 - ▶ Dependen de la implementación o microarquitectura.
- ▶ Un ISA es un modelo independiente de cuestiones de temporización.
 - ▶ Si un programa respeta las especificaciones del ISA, su ejecución es garantizada como correcta, en cualquier implementación.

Repaso: programas de usuario

- ▶ Usan instrucciones no privilegiadas del ISA.
 - ▶ El SO usa las instrucciones privilegiadas.
- ▶ Mantienen un estado cuando están en ejecución.
- ▶ Se ejecutan normalmente en un entorno de memoria virtual.
 - ▶ El SO maneja las tablas de páginas de cada programa.
 - ▶ La MMU se encarga de la traducción de direcciones virtuales a direcciones físicas.
 - ▶ Las entradas de la tablas de páginas contienen bits de protección.
- ▶ Solicitan servicios al SO a través de llamadas al sistema.
 - ▶ El SO se encarga de cambiar de proceso (y guardar el estado).

El problema “nuevo”

- ▶ Hasta aquí, todas las técnicas vistas en Arquitectura de Computadoras hacen que una computadora funcione completamente bien.
 - ▶ Siempre más rápido, con mejor performance.
 - ▶ Desde hace 20 años controlando el consumo de energía de manera eficiente.
 - ▶ “Las tres P”: *Price, Performance, Power*.
- ▶ Pero, *¿es posible que aunque los programas funcionen correctamente, se filtre información?*
 - ▶ Sí, mediante **ataques tipo *side-channel***.

Ataques tipo *side-channel*

- ▶ Ataques basados en **mediciones físicas indirectas** de un sistema informático.
 - ▶ No buscan debilidades inherentes del sistema, sino canales que no deberían ser usados para transmitir información.
 - ▶ La operación “normal” de un sistema produce ciertas variables físicas **secundarias** que pueden ser medidas, y de esa manera deducir información sobre el sistema mismo.
- ▶ Ejemplos:
 - ▶ Mediciones de duración de ciertas operaciones.
 - ▶ Mediciones de consumo de energía.
 - ▶ Mediciones de sonidos producidos por el sistema.
 - ▶ Mediciones de tiempos de tipeo. ¡También de sonidos al tipear!
 - ▶ SATAn: usa cables SATA como antenas para transferir señales de radio en la frecuencia de 6 GHz.
 - ▶ Mediciones de radiaciones electromagnéticas en cables HDMI (Deep-Tempest, 2024) y en módulos RAM (RAMBO, 2024).

Ataques tipo *side-channel* a los cachés

- ▶ A mediados de 2017, por ocurrencia de algunos investigadores.
- ▶ Artículos revelados al público en Enero de 2018.
- ▶ La jerarquía de memoria altera la velocidad de acceso a un dato.
 - ▶ El dato se accede más rápido cuanto más cerca del procesador esté.
- ▶ Midiendo el tiempo que demora un cierto acceso a memoria, es posible determinar en cuál caché se ubica.
 - ▶ El tiempo de ejecución de un determinado programa variará según su ubicación en los cachés de la jerarquía de memoria.

Listado de vulnerabilidades 2018-2021

- ▶ **Meltdown** (Rogue Data Cache Load)
- ▶ **Spectre-v1** (Bounds Check Bypass)
- ▶ **Spectre-v2** (Branch Target Injection)
- ▶ Spectre-v3a (Rogue System Register Read)
- ▶ Spectre-v4 (Speculative Store Bypass)
- ▶ BranchScope (Directional Predictor Attack)
- ▶ Spectre-v1.1 (Bounds Check Bypass Store)
- ▶ Spectre-v1.2 (Read-only Protection Bypass)
- ▶ LazyFPU save/restore
- ▶ TLBleed (TLB side-channel attacks)
- ▶ SpectreRSB (Return Predictor Attack)
- ▶ NetSpectre (Spectre over the network)
- ▶ Foreshadow (L1 Terminal Fault)
- ▶ PortSmash (Port Contention between Threads)
- ▶ SPOILER (ASLR attack against disambiguation)
- ▶ MDS (Microarchitectural Data Sampling)
- ▶ ZombieLoad (Cross Privilege-Boundary Data Leakage)
- ▶ RIDL (Rogue In-flight Data Load)
- ▶ FallOut
- ▶ Medusa
- ▶ SMOtherSpectre
- ▶ SWAPGS
- ▶ Ryzenfall
- ▶ Chimera
- ▶ Masterkey
- ▶ TAA (TSX Asynchronous Abort)
- ▶ L1DES (L1D Eviction Sampling)
- ▶ VRS (Vector Register Sampling)
- ▶ CacheOut
- ▶ NetCAT (Network Cache Attack)
- ▶ LSGAxe
- ▶ Snoop (Snoop-Assisted L1 Data Sampling)
- ▶ LVI (Load Value Injection)
- ▶ Take-A-Way (L1D Way Predictor)
- ▶ CrossTalk (Special Register Buffer Data Sampling)
- ▶ SLS (Straight Line Speculation)
- ▶ Rowhammer
- ▶ RAMBleed
- ▶ PlunderVolt
- ▶ PSF (Predictive Store Forwarding)
- ▶ Dead uOps
- ▶ M1RACLES

Listado de vulnerabilidades 2021-2026

- ▶ Augury
 - ▶ Blindside
 - ▶ PACMAN
 - ▶ Hertzbleed
 - ▶ Retbleed
 - ▶ AMD Prefetch Attacks
 - ▶ Binoculars
 - ▶ TLB;DR
 - ▶ Half-Double
 - ▶ AEpic Leak
 - ▶ SQUIP
 - ▶ SATAn
 - ▶ Hot Pixels
 - ▶ Zenbleed
 - ▶ Downfall
 - ▶ Inception
 - ▶ Phantom
 - ▶ CacheWarp
 - ▶ Reptar
 - ▶ SLAM
 - ▶ LeftoverLocals
 - ▶ TIKTAG (ARM)
 - ▶ Indirector
 - ▶ GhostWrite (RISC-V)
 - ▶ GhostRace
 - ▶ SledgeHammer
 - ▶ ZenHammer
 - ▶ Pixel Thief
 - ▶ Peep with a Mirror
 - ▶ Heckler
 - ▶ RAMBO
 - ▶ Deep-Tempest
 - ▶ BadRAM
 - ▶ EntrySign (AMD)
 - ▶ SLAP (Apple)
 - ▶ FLOP (Apple)
 - ▶ Branch Privilege Injection
 - ▶ Training Solo
 - ▶ GPUHammer (Nvidia, Jul-25)
 - ▶ TRRespass
 - ▶ Phoenix (Sep-25)
- ▶ En promedio, se descubre una nueva vulnerabilidad cada 30 días.
 - ▶ Algunas buscan nombres originales y pegadizos. Y tienen logos.
 - ▶ Algunas afectan exclusivamente a un fabricante, o a todos menos a uno.
 - ▶ No se tiene certeza que alguna haya sido efectivamente explotada.

Soluciones a las vulnerabilidades

- ▶ La mayoría no puede ser solucionada de manera sencilla.
 - ▶ Porque afectan cuestiones inherentes al diseño de los procesadores modernos.
 - ▶ Cachés, TLB, buffers, predicción de saltos, DVFS, redes de interconexión, el controlador de interrupciones, GPUs, etc.
- ▶ Por suerte existen “paliativos” (*mitigations*).
 - ▶ Ya sea por hardware o por software.
 - ▶ Reducen significativamente el posible impacto de la vulnerabilidad.
- ▶ ¡A costa de una reducción de performance!
 - ▶ Entre un 2% y un **35%**, dependiendo de la aplicación, del procesador y de la vulnerabilidad, con una media en una reducción del 10%.
 - ▶ Representan un **nuevo compromiso entre seguridad y performance**.
- ▶ Intel comenzó en 2019 un programa de recompensas por bugs encontrados en sus procesadores.
 - ▶ También patrocina a los que encuentran bugs en la competencia 😊
 - ▶ AMD hizo el suyo abierto al público en general el 30/05/24 😊
 - ▶ Google también tiene el suyo.

Limitaciones a los paliativos

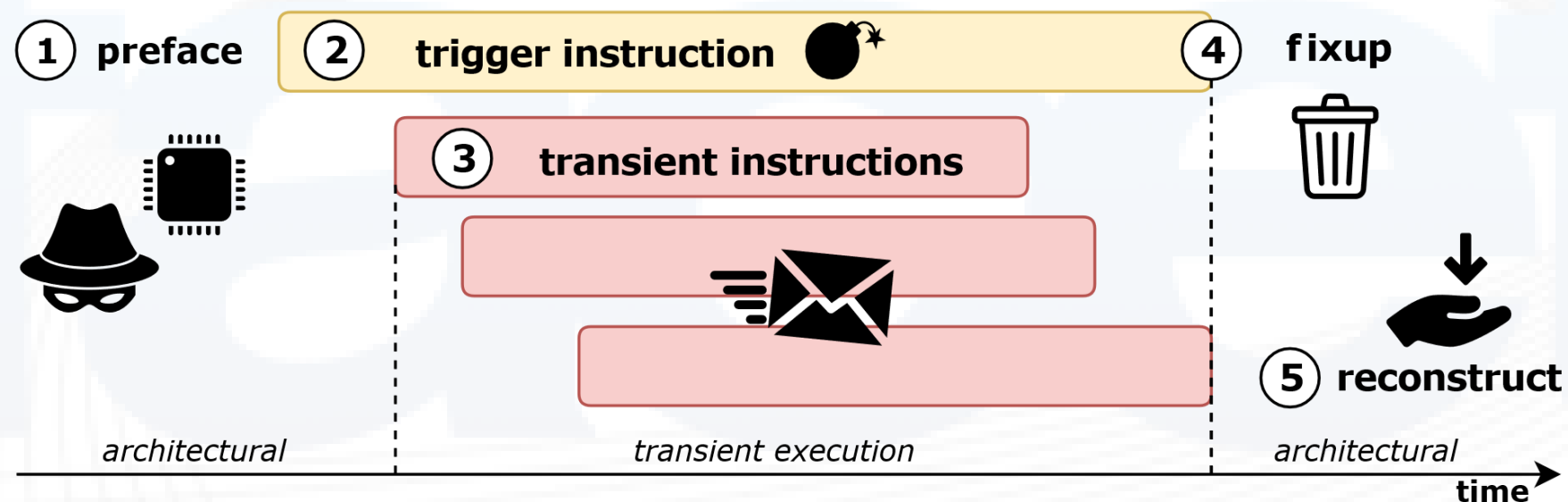
- ▶ El hardware no puede ser “reemplazado in situ”.
- ▶ Los paliativos no siempre son posibles y no siempre son razonables.
 - ▶ La mayoría de las veces son tareas difíciles y que consumen mucho tiempo de ingeniería.
- ▶ Muchos implican deshabilitar ciertas características de los procesadores.
 - ▶ Algunos son obligatorios, otros opcionales.
 - ▶ Algunos son simples de incorporar en hardware a futuro.
- ▶ Algunos paliativos son basados en nuevas instrucciones / microinstrucciones.
 - ▶ Agregados a un ISA incremental, complejo.
 - ▶ Implican reescribir/recompilar/redistribuir software existente.
- ▶ En muchos casos, es el SO el que aparece para ayudar.
 - ▶ Porque es mucho más simple de modificar (sw vs hw).
 - ▶ Y más “genérico” (soluciona errores de varios procesadores).

Un ejemplo de paliativo

- ▶ Muchos de los ataques son contra los cachés, sobre todo contra los LLC.
 - ▶ *¿Por qué no contra los L1 o L2?*
- ▶ Paliativo: **cachés randomizados**.
 - ▶ El mapeo de direcciones ya no es directo, sino que es a través de una función *hash* con una semilla aleatoria.
 - ▶ Genera la abstracción que el caché es asociativo puro, con política de reemplazo random.
 - ▶ Algoritmo MIRAGE, presentado en Ago-21. Mejorado por Maya, presentado en Jul-24 (en CABA).
 - ▶ Ventaja: elimina muchos de los posibles ataques.
 - ▶ Desventaja: hace más lento y más costoso al LLC.
- ▶ Otro: **cachés particionados**.
 - ▶ Cada programa recibe un pedazo de caché que no es compartido con otros programas.
 - ▶ Ventaja: completo aislamiento de los datos.
 - ▶ Desventaja: disminución significativa de performance, sobre todo al incrementar la cantidad de núcleos.

Clasificación de los ataques (1/2)

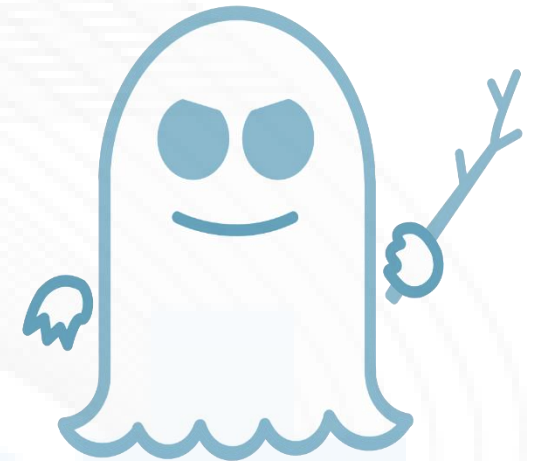
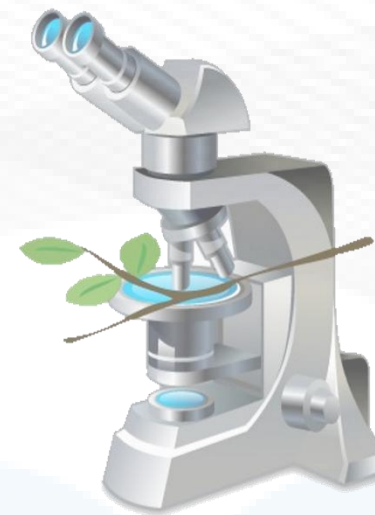
- ▶ Los ataques más comunes son los llamados **Ataques de Ejecución Transitoria**.
 - ▶ **¿Recuerdan las excepciones precisas? ¿Y cuándo se las evaluaba?**
 - ▶ Durante un breve período de tiempo se ejecutan instrucciones que modifican algún estado de la microarquitectura.
 - ▶ Esas instrucciones son eventualmente descartadas a nivel arquitectura, pero los rastros que dejan no, dando lugar a posibles *side-channels*.



- ▶ Introducen **un nuevo tipo de dependencia: por seguridad**.
- ▶ Más info en: <https://transient.fail/>
- ▶ A su vez, se dividen en dos grandes tipos, según la causa por la que se ingresa a la ejecución transitoria.

Ataques de Ejecución Transitoria

- ▶ Ataques tipo **Meltdown**
 - ▶ Relacionados con romper barreras de aislamiento.
 - ▶ Ocasionados por **excepciones**, de cualquier tipo.
 - ▶ Saltean políticas de seguridad basadas en hardware
- ▶ Ataques tipo **Spectre**
 - ▶ Todos los relacionados con ejecutar código de manera especulativa.
 - ▶ Ocasionados por **fallas en la predicción** del flujo de datos o de control.
 - ▶ Saltean políticas de seguridad basadas en software.
- ▶ Combinaciones de ambos.



Meltdown (*Rogue Data Cache Load*)



MELTDOWN

► ¿Qué está mal en este código?

```
if (condición que haga fallar la predicción de saltos) {  
    trusted_value = kernel_mem[untrusted_offset];  
    tmp = user_mem[(trusted_value) & mask];  
}
```

- Mientras se resuelve la condición del *if*, las instrucciones dentro del cuerpo se ejecutan **especulativamente**.
 - En la primera línea dentro del *if* hay una instrucción Ld que intenta acceder a una dirección de memoria no permitida.
 - Como no se tienen los permisos, genera una excepción. Pero por ser especulativa, no se le da importancia hasta llegar al *commit*.
 - Se trae igual el dato, para ganar tiempo, por las dudas. Al finalizar la especulación, el dato será descartado.
 - Pero mientras tanto, ese dato es usado como índice de una segunda instrucción Ld, para acceder a otra dirección de memoria que sí es conocida y permitida. Pero este nuevo valor también será descartado.

Meltdown (*Rogue Data Cache Load*)



MELTDOWN

- ▶ Entonces, a fin de cuentas, los dos valores leídos de memoria serán descartados, y ninguno de los dos serán accesibles. *¿Cuál es el problema entonces?*
- ▶ Que deja rastros en los cachés, porque estos sí fueron cargados (por las dudas).
- ▶ Luego, con otro programa, se miden los tiempos de acceso a esa memoria conocida.
 - ▶ El acceso que demore menos... ¡será el correspondiente al dato traído especulativamente!
 - ▶ Así, es posible conocer el *trusted_value*, sin haber accedido nunca al mismo.

Meltdown – Paliativos

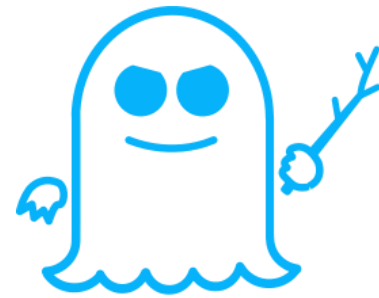
- ▶ Afectó a todos los procesadores x86 de Intel y los ARMv8, con ejecución fuera de orden y ejecución especulativa.
 - ▶ No a los de AMD.
 - ▶ Hasta que lograron romperlos con AMD Prefetch Attacks (en 2022)
- ▶ El paliativo más simple implica separar totalmente las tablas de páginas de los programas de usuario y del SO.
 - ▶ *Kernel Page Table Isolation (KPTI)*.
 - ▶ Muchos más retardos en cada llamada/retorno al SO.
 - ▶ Vaciar los TLB en cada llamada/retorno al SO.
- ▶ Los procesadores que usan identificadores de proceso en los TLB (ASID) fueron mucho menos afectados.

Ejemplos adicionales



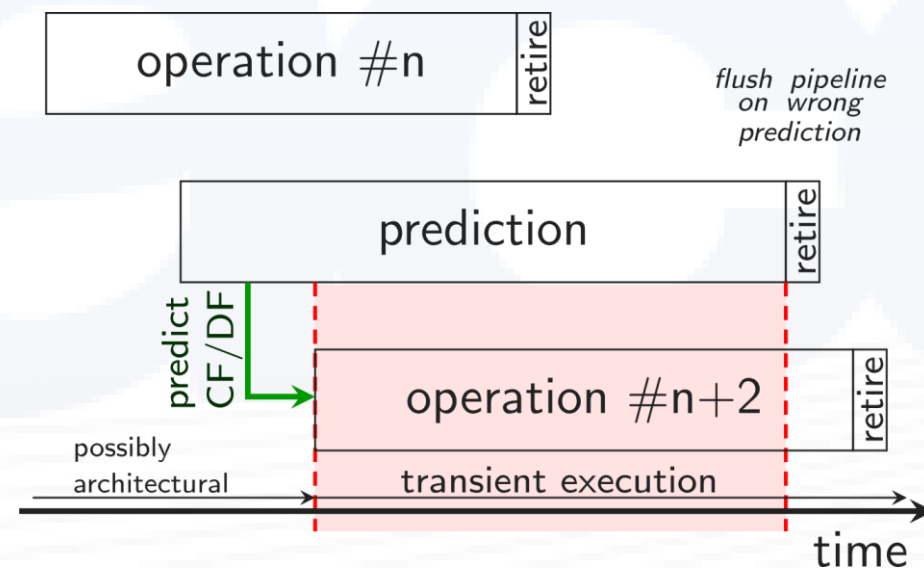
- ▶ **Foreshadow (L1 Terminal Fault).**
 - ▶ Normalmente los cachés L1 son VIPT para permitir su acceso en paralelo con TLB.
 - ▶ La comparación de etiquetas físicas implica una traducción exitosa por parte del TLB de DV a DF.
 - ▶ Y también implica que se cuenta con los permisos necesarios en la entrada del TLB (p.ej: validez).
 - ▶ Para ganar tiempo, Intel hacía que sus procesadores enviaran la dirección física al L1 antes de verificar si la entrada era válida o no 😊
- ▶ **ZombieLoad, PortSmash y SQUIP.**
 - ▶ En SMT, los *threads* independientes comparten recursos. En particular, la jerarquía de memorias es totalmente compartida.
 - ▶ Revisan continuamente los accesos a memoria provocados por otro *thread* dentro del mismo núcleo y van muestreando datos. Eventualmente, serán datos interesantes.
 - ▶ **En algunos casos, se recomienda desactivar el uso de *multithreading* 😊**

Spectre



SPECTRE

- ▶ Junto con Meltdown, los primeros en aparecer.
- ▶ No es una vulnerabilidad en sí, sino una familia de vulnerabilidades.
 - ▶ Todas relacionadas con ejecución **especulativa** en procesadores con ejecución fuera de orden.
- ▶ Basado en ejecutar “sin querer” código existente que no debería ser ejecutado.
 - ▶ Por ejemplo, porciones del kernel del SO, o leer datos privados de memoria protegida.



- ▶ Hace que cualquier aplicación sea vulnerable.
 - ▶ Inclusive aquellas que siguen las prácticas de seguridad recomendadas.

Spectre v1 (*Bounds Check Bypass*)



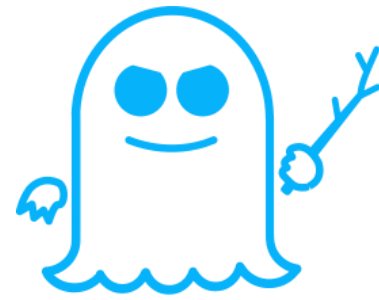
SPECTRE

► ¿Qué está mal en este código?

```
if (untrusted_offset < array_length) {  
    trusted_value = private_mem[untrusted_offset];  
    tmp = other_data[(trusted_value) & mask];  
    ...  
}
```

- En un primer vistazo... nada. No hay ninguna excepción, como en Meltdown.
- Sin embargo, permite que se ejecute código especulativamente aunque no se cumpla un chequeo de límites.
- Se usa el resultado de una búsqueda en un array como índice de búsqueda en un segundo array.
- Mediante mediciones indirectas de los tiempos de acceso de caché del segundo array, es posible obtener el valor de *trusted_value*, porque será notoriamente más rápido que los demás accesos.

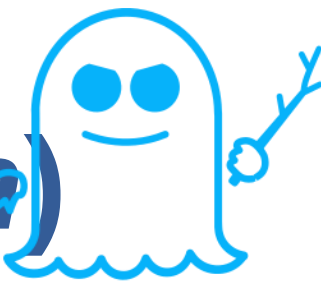
Spectre v1 – Paliativos



SPECTRE

- ▶ **No se puede corregir por hardware 😊**
 - ▶ Simplemente, habría que deshabilitar la ejecución especulativa.
 - ▶ O diseñar una arquitectura completamente nueva.
- ▶ Se puede modificar el programa para agregar instrucciones que eviten la ejecución especulativa mediante “barreras”.
 - ▶ Una instrucción LFENCE antes de la instrucción LOAD.
- ▶ Se desarrollaron herramientas que detectan secuencias peligrosas en los programas, para ayudar a los desarrolladores a modificar sus códigos.
- ▶ Para los más curiosos: pueden probar una demo en <https://leaky.page/>

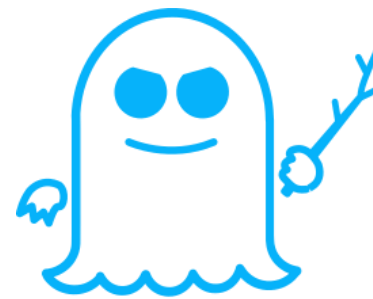
Spectre v2 (*Branch Target Injection*)



SPECTRE

- ▶ Los predictores de salto se basan en buffers de historial (BHT) y de destino (BTB).
 - ▶ Estos buffers son compartidos.
 - ▶ Es posible “entrenar” a los predictores para saltar a algún lugar deseado por el atacante.
 - ▶ Aprovechándose que los predictores de salto se manejan con direcciones virtuales.
 - ▶ Hasta que la falla en la predicción es detectada, se ejecutan algunas instrucciones especulativamente.
 - ▶ Estas instrucciones modifican algún estado, con un par de instrucciones similar a la variante anterior.
 - ▶ Y luego, obtener datos mediante mediciones de tiempos de acceso a los cachés a través de *side-channels*.

Spectre v2 – Paliativos



SPECTRE

- ▶ **Deshabilitar predicción de saltos 😊**
- ▶ Borrar el historial del predictor en cada cambio de contexto.
- ▶ x86 definió nuevas instrucciones que sirven para mitigar los efectos.
 - ▶ Pueden llegar a costar miles de ciclos.
- ▶ Google inventó una solución 100% software.
 - ▶ Eliminar todos los saltos indirectos (**jalr** y **ret** en RV32I) y reemplazarlos por una función que modifique el stack poniendo el destino deseado.
 - ▶ “Trampolines de retorno” (*retpolines*).
 - ▶ Los cuales fueron vulnerados en 2022 con Retbleed 😞
- ▶ Los fabricantes de procesadores están continuamente implementando diversos tipos de paliativos en hw.

Spectre – Otras variantes

- ▶ Muchas otras variantes más aparecieron:
 - ▶ Todas vinculadas a ejecutar de manera especulativa pedazos de código que modifiquen distintos buffers de estado.
 - ▶ Y luego usar técnicas de side-channel sobre los buffers para obtener los valores.
 - ▶ **TLBleed** es un ataque sobre el TLB, cuando se usa multithreading.
 - ▶ **NetSpectre**, usa buffers usados en transmisiones por una red.
 - ▶ **MDS**, varias vulnerabilidades divulgadas que atacan buffers internos no documentados.
 - ▶ **LOTR**, que ataca los buses de interconexión *on-chip*.
 - ▶ **SLAM**, que ataca todo el mecanismo de traducción de direcciones virtuales.
- ▶ Y seguramente muchas más seguirán apareciendo.
- ▶ **Cualquier recurso de un procesador que pueda ser compartido, puede convertirse en un potencial candidato a un ataque *side-channel* de tipo Spectre.**
- ▶ **Cualquier paliativo de ataques *side-channel* inevitablemente empeora la performance y aumenta el costo.**

Phantom e Inception (2023)

- ▶ Los procesadores más recientes de Intel y AMD empiezan a especular “demasiado pronto”, inclusive antes de decodificar una instrucción.
 - ▶ La predicción realizada por los *prefetchers* vistos en el Tema 11.
 - ▶ Ya no solo se habla de ejecución transitoria, sino también de decodificación transitoria y de fetch transitorio.
 - ▶ Permiten abrir ventanas de especulación prácticamente desde cualquier instrucción (**Phantom**).
 - ▶ O sea, por ejemplo, hacer que una instrucción `add` actúe como un salto.
- ▶ Por otra parte, todos los ataques tipo Spectre buscan leer datos escritos de manera especulativa.
 - ▶ Hasta que a alguien se le ocurrió no leer datos, sino usar la ventana especulativa para entrenar incorrectamente la predicción de saltos.
 - ▶ De esta manera, generan **futuras** ventanas de ejecución transitoria; una especie de Spectre “recursivo”.
 - ▶ Finalmente, es posible que un programa termine entrenándose a si mismo a fallar la predicción de sus propios saltos (**Inception**).

Clasificación de los ataques (2/2)

- ▶ Un segundo tipo de ataque son los llamados **Ataques de Reutilización de Código**.
 - ▶ La idea es componer un código malicioso juntando pedacitos de código de librerías existentes, que sirvan para desactivar las protecciones de memoria.
 - ▶ Ataques Return-Oriented Programming (mediante instrucciones de retorno).
 - ▶ Ataques Jump-Oriented Programming (mediante saltos indirectos).
 - ▶ Ataques Block-Oriented Programming (usan bloques básicos de programas existentes).
 - ▶ Aplicables a GPU también.
- ▶ El tercer tipo de ataque más común son los **Ataques estilo Rowhammer**.

Ataques estilo Rowhammer

- ▶ Orientados a fallos relacionados a la tecnología de los componentes.
- ▶ Funciona por interferencia electromagnética, porque en los últimos procesos de fabricación, las celdas están muy cercanas entre sí.
- ▶ Es posible modificar bits en RAM de manera predecible, sin accederlos.
 - ▶ Realizando repetidamente accesos de lecturas a las filas adyacentes.
- ▶ Así es posible modificar permisos en las tablas de páginas, entre otros.

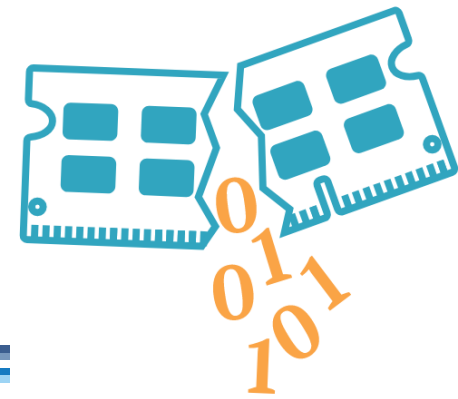
Ataques estilo Rowhammer

- ▶ Alcanza a más del 80% de los chips de DRAM disponibles en el mercado.
 - ▶ Inclusive, hasta DDR5 (denominado Phoenix).
- ▶ Pueden probarlo uds. mismos:
 - ▶ <https://github.com/google/rowhammer-test>
 - ▶ <https://github.com/IAIK/rowhammerjs> (se ejecuta en un navegador web).
- ▶ Múltiples versiones:
 - ▶ En JavaScript (RowhammerJS, 2016).
 - ▶ Para dispositivos móviles con Android (Drammer, 2016), incluso utilizando el GPU para hacerlo más rápido.
 - ▶ A través de la red (Throwhammer y Nethammer, ambos del 2018).
 - ▶ Half-Double (del 2022) ataca celdas cercanas pero no adyacentes ☹
 - ▶ SledgeHammer (2024), mejora el ataque aplicando paralelismo.
 - ▶ GPUHammer (2025), ataca las memorias GDDR de placas Nvidia.
 - ▶ ¡Atacan también las memorias FLASH!

Ataques estilo Rowhammer – Paliativos

- ▶ Limitar la cantidad de accesos a una misma fila (reduce performance).
- ▶ Refrescar las filas con mayor frecuencia (reduce performance y aumenta consumo de energía).
- ▶ Agregar correcciones de errores en los chips de DRAM (aumenta el costo y el consumo de energía).
- ▶ Al acceder a una fila, refrescar una adyacente con una cierta probabilidad (aumenta un poco la complejidad).
- ▶ Las memorias RAM no volátiles pueden representar una solución diferente.

RAMBleed



RAMBleed

- ▶ Presentado en mayo de 2020.
- ▶ Combinación de ataque estilo Rowhammer con ataque *side-channel*.
- ▶ Permite **leer** bits de memoria física, sin accederlos.
 - ▶ Inclusive de otros programas.
 - ▶ Usa Rowhammer no para escribir, sino para leer.
 - ▶ Un bit es más propenso a cambiar su valor, si los bits de las filas adyacentes tienen el valor opuesto.
- ▶ Gran problema de confidencialidad, ya que deja de haber datos “secretos”.
 - ▶ En el caso ejemplo, extrajeron una clave SSH encriptada de 2048 bits.
- ▶ No es mitigado por las técnicas de corrección de errores más comunes.

Zonas de Ejecución Segura

- ▶ Con el objetivo de reforzar aún más la seguridad de los procesadores, ahora también incorporan unidades de manejo de seguridad.
 - ▶ A la larga, son como nuevos niveles de privilegio.
 - ▶ Intel SGX (Software Guard eXtension), AMD SEV (*Secure Encrypted Virtualization*) y SME (*Security Memory Encryption*), ARM TrustZone y Shield para RISC-V.
- ▶ Microsoft diseñó en 2020 un núcleo específicamente diseñado para proveer seguridad: Pluton.
 - ▶ Incorporado por AMD (como un *chiplet*) en todos sus procesadores a partir de **ene-2022**.
- ▶ Estos coprocesadores no pueden ser accedidos por máquinas virtuales.
 - ▶ Representan lo que se conoce como un *Hardware Root-of-Trust*.
 - ▶ Deben estar habilitados por el SO y/o capa de virtualización que esté por encima.
- ▶ Buscan evitar ataques físicos a la RAM principal.

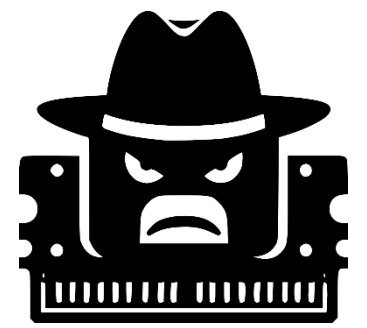
Zonas de Ejecución Segura

- ▶ Proveen zonas de ejecución seguras (TEE, *Trusted Execution Environments*)
 - ▶ Cómo bóvedas de seguridad dentro de los procesadores.
 - ▶ El contenido de esas bóvedas está protegido, y no puede ser ni accedido ni modificado desde fuera de la bóveda.
- ▶ Incorporan funciones de encriptación en hardware.
 - ▶ Todos los accesos a memoria son encriptados. En especial, al momento de booteo.
 - ▶ Las claves de acceso son generadas internamente por estas unidades.
 - ▶ Inclusive, pueden incorporar circuitos de autodestrucción en caso de resultar comprometidos.
- ▶ Proveen un total aislamiento de los programas.
 - ▶ Cada memoria virtual es absolutamente independiente de las demás.
 - ▶ Obligatorio para máquinas virtuales que se ejecutan “en la nube”.
 - ▶ Idea que vino desde las consolas de videojuegos 😊

PlunderVolt (y otros)

- ▶ En vez de modificar bits en memoria, los modifica dentro del procesador.
- ▶ Los procesadores que implementan DVFS suelen poseer instrucciones para gestionar esta característica.
 - ▶ Son instrucciones privilegiadas, no accesibles a programas de usuario.
- ▶ La idea básica consiste en disminuir la tensión de alimentación hasta que detecta que empiezan a ocurrir las fallas.
 - ▶ Induciendo fallas en las instrucciones que escriben datos en las bóvedas de seguridad, es posible que estos datos se escriban fuera de las mismas, haciéndolos accesibles.
 - ▶ Por lo tanto, necesita tener privilegios de root.
- ▶ Ataca en particular al SGX de los procesadores de Intel.
 - ▶ Permite extraer claves criptográficas, mediante otro *side-channel*.
 - ▶ CLKScrew y VoltJockey son similares, pero atacan la TrustZone de ARM.
 - ▶ Por suerte, pueden mitigarse fácilmente por software.
- ▶ CacheWarp utiliza el protocolo de coherencia de los cachés para acceder a la SEV en los procesadores de AMD.
- ▶ Heckler usa vectores de interrupción para inyectar códigos maliciosos y romper la integridad y confidencialidad de estas barreras.

BadRAM



- ▶ Rompe la zona de ejecución segura de AMD (SEV).
- ▶ Usando una Raspberry Pi Pico, hacen que un módulo de DRAM le mienta su tamaño al procesador.
 - ▶ Por ejemplo, duplicando su tamaño.
 - ▶ Esto genera direcciones de memoria fantasma, que son alias de las direcciones reales.
- ▶ Mientras las direcciones de memoria reales están totalmente protegidas... los alias no.
- ▶ Requiere acceso físico a los módulos de DRAM.
- ▶ AMD ya liberó parches de firmware que lo corrigen.

Y por si fuera poco... hay SDCs

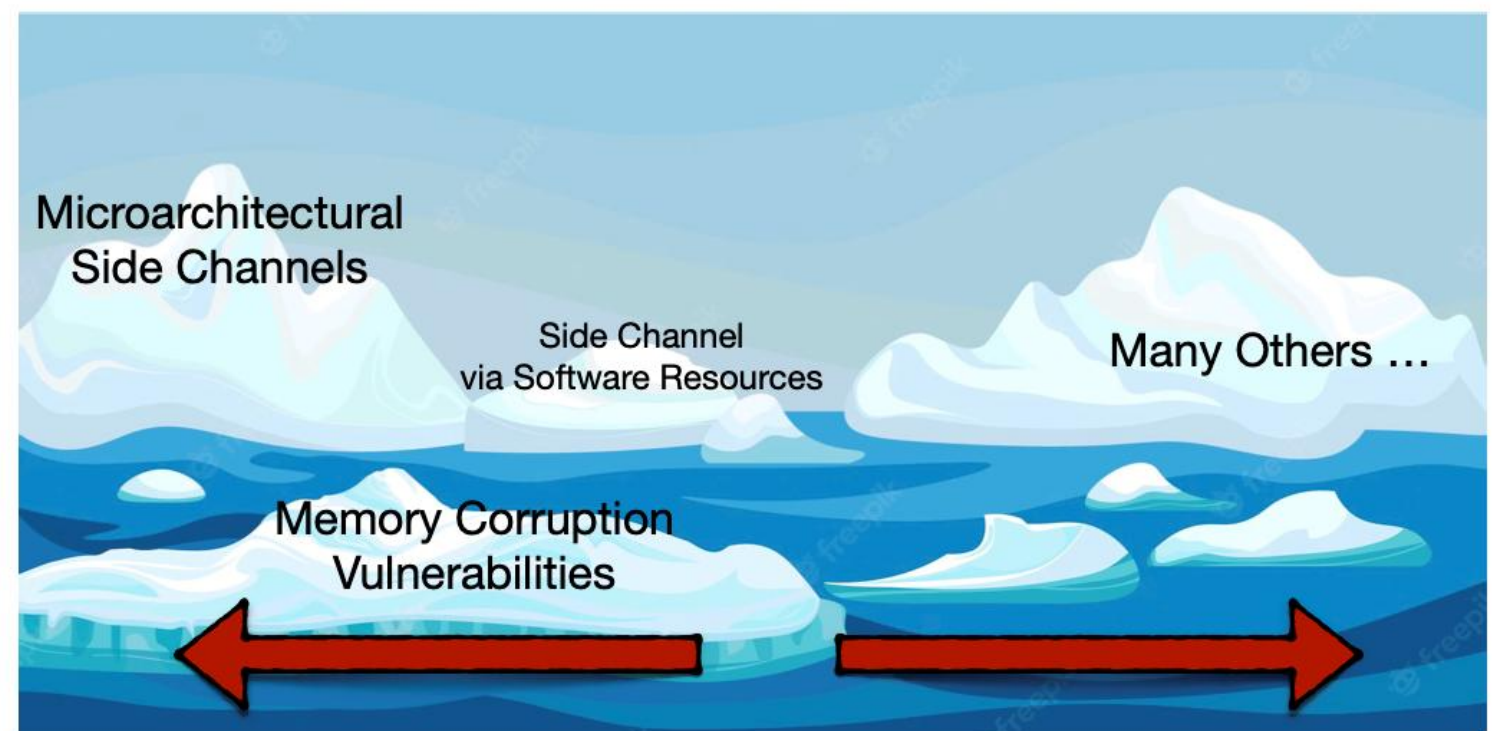
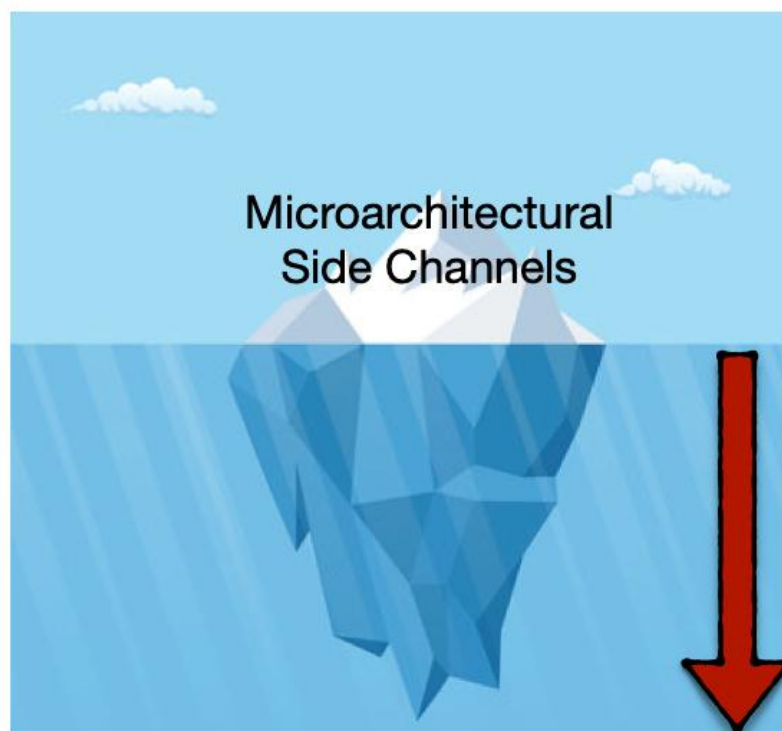
- ▶ Google y Meta reportaron en 2021 (en Junio y en Febrero, respectivamente) que hay núcleos que simplemente... calculan mal.
 - ▶ Alibaba y Nvidia confirmaron los datos en 2023.
- ▶ En principio, son errores **inclasificables, indetectables e incorregibles**.
 - ▶ Denominados “Corrupción de datos silenciosa” (*Silent Data Corruption*).
 - ▶ Son raros (aprox 1 cada 3000 chips). Pero más comunes de lo que se esperaría.
 - ▶ GPUs, TPUs y otros DSA son igualmente afectados.
- ▶ Se sabe que no son debido a errores de diseño.
 - ▶ Pero no se sabe con certeza (aún) por qué ocurren 😊
 - ▶ Posibles causas identificadas: errores de fabricación que pasaron los tests, errores por envejecimiento, o simplemente por variabilidades de timing.
 - ▶ Y por lo tanto, no existen herramientas para detectarlos y verificarlos.

Y por si fuera poco... hay SDCs

- ▶ Pasaron a ser notorios porque este tipo de empresas empezaron a tener instalaciones con ~100k servidores, con ~1M núcleos.
 - ▶ Se hicieron más notorios con cada nueva IA que aparece, porque su entrenamiento sufre mucho estos problemas.
- ▶ Existen esfuerzos conjuntos de la industria y de la academia para encarar esta problemática.
 - ▶ El objetivo actualmente consiste en minimizar su impacto.
- ▶ Si la causa es que se está llegando a los límites de la escalabilidad en el proceso de fabricación, puede representar un nuevo compromiso entre costo/performance y confiabilidad.
 - ▶ *¿Habrá que agregar redundancia a los cores? ¿O técnicas de corrección de errores? ¿Nuevos métodos de testeo? ¿Nuevas herramientas de simulación?*

El verdadero inconveniente

- ▶ Todas las vulnerabilidades descubiertas no son el problema real.
 - ▶ Al fin y al cabo, nadie las explotó aún (al menos que se sepa).
- ▶ Tampoco lo son todas las nuevas vulnerabilidades a descubrir.
- ▶ El problema real es que se abrió una “caja de Pandora” que se desconocía, y que afectará todos los diseños futuros.



¿Otro nuevo paradigma?

- ▶ La performance debe seguir mejorando, pero ahora en un mundo “*post Meltdown*”.
- ▶ Las mejoras de performance suelen provenir de tres lugares: mejor proceso de fabricación, mejoras en el diseño de la microarquitectura, mejor adaptación a nuevos workloads.
 - ▶ Los dos últimos generarán nuevos riesgos de seguridad
- ▶ Se inventarán nuevas técnicas que solucionen de algún modo todos los problemas conocidos.
- ▶ Pero se continuarán descubriendo nuevos problemas.
- ▶ A medida que se diseñen nuevas arquitecturas para mejorar la performance (DSA) es muy probable que se introduzcan nuevas vulnerabilidades que hoy aún no existen.

¿Otro nuevo paradigma?

- ▶ Los ataques de ejecución transitoria muestran que la seguridad debe ser incorporada desde los niveles más bajos de diseño, porque si no, todo lo que se defina por encima (el ISA) puede romperse.
- ▶ Los ataques de tipo Rowhammer y de Corrupción de datos silenciosa muestran que el hardware es inherentemente falible.
- ▶ Es necesario cambiar la manera en la que los procesadores son diseñados actualmente.
 - ▶ No solamente para paliar vulnerabilidades encontradas.
 - ▶ Sino para **agregar la seguridad como un aspecto fundamental de diseño** que compromete la performance.
 - ▶ Al igual que el costo y el consumo de energía.

Preguntas sin respuestas (aún)

- ▶ *¿Cuánto es la cantidad adecuada de seguridad que debería incluir un diseño?*
 - ▶ Las soluciones propuestas idealmente deberían poder evolucionar con el tiempo, para adaptarse a nuevos ataques que se descubran.
- ▶ *¿Habrá que crear benchmarks de seguridad?*
 - ▶ No todas las aplicaciones tienen los mismos requerimientos de seguridad.
- ▶ *¿Es necesario redefinir los ISA actuales?*
 - ▶ Queda demostrado que el estado de un microprocesador no es solamente su interfaz visible al programador.
 - ▶ Hacer que las implementaciones consideren ciertos aspectos dependientes del tiempo (transitorios), y así hacerlos menos susceptibles a ataques *side-channel*.
 - ▶ O extenderlos para que incluyan nuevas características de seguridad.
- ▶ *¿Se podrá obtener la misma performance actual sin utilizar ejecución especulativa?*
 - ▶ Quizás las Arquitecturas de Dominio Específico puedan ayudar en este aspecto.

Resumen final

- ▶ Ataques tipo *side-channel*: basados en mediciones físicas indirectas.
 - ▶ Tiempos de acceso a un dato en cachés.
- ▶ Muchas vulnerabilidades descubiertas desde enero de 2018.
 - ▶ Y muchas más por descubrir aún.
 - ▶ No se sabe que hayan sido explotadas (por suerte).
 - ▶ La mayoría no tiene solución. Sólo paliativos.
 - ▶ Explotan algún tipo de buffer existente en el diseño.
 - ▶ Ataques tipo Meltdown y ataques tipo Spectre.
- ▶ Los paliativos presentan un nuevo compromiso
 - ▶ Más seguridad = menos performance.
 - ▶ También presentan problemas y limitaciones.
- ▶ Seguridad debería ser un criterio fundamental en Arquitectura de Computadoras.
 - ▶ No considerada durante 50 años.

Referencias

- ▶ “After Meltdown and Spectre: Security Concerns Facing Contemporary Microarchitectures”, presentation by Jon Masters, Red Hat (Abr-2019).
- ▶ “On the Meltdown & Spectre Design Flaws” presentation by Mark Hill (Feb-2018).
- ▶ “Spectre is here to stay: An analysis of side-channels and speculative execution”, Google (Feb-2019).
- ▶ “On the Spectre and Meltdown Processor Security Vulnerabilities”, Hill et al, IEEE Hot Chips (Abr-2019).
- ▶ “Hot Pixels: Frequency, Power, and Temperature Attacks on GPUs and Arm SoCs”, Taneja et al (May-2022).
- ▶ “AMD Prefetch Attacks through Power and Time”, Lipp et al (Ago-2022).
- ▶ “Binoculars: Contention-Based Side-Channel Attacks Exploiting the Page Walker”, Zhao et al (Ago-2022).
- ▶ “Half-Double: Hammering From the Next Row Over”, Kogler et al. (Ago-2022).
- ▶ “A Practical Deep Learning-Based Acoustic Side-Channel Attack on Keyboards”, Harrison et al. (Ago-2023).
- ▶ “Leaky Address Masking: Exploiting Unmasked Spectre Gadgets with Noncanonical Address Translation”, Hertogh et al. (Dic-2023).
- ▶ “Phantom: Exploiting Decoder-detectable Mispredictions”, Wikner et al (Oct-2023)
- ▶ “INCEPTION: Exposing New Attack Surfaces with Training in Transient Execution”, Trujillo et al (Ago-2023).
- ▶ “TIKTAG: Breaking ARM’s Memory Tagging Extension with Speculative Execution”, Kim et al (Jun-2024).
- ▶ “RAMBO: Leaking Secrets from Air-Gap Computers by Spelling Covert Radio Signals from Computer RAM”, Guri (Sep-2024).

Referencias

- ▶ <https://meltdownattack.com/>
- ▶ <https://foreshadowattack.eu/>
- ▶ <https://zombieloadattack.com/>
- ▶ <https://mdsattacks.com/>
- ▶ <https://viattack.eu/>
- ▶ <https://plundervolt.com/>
- ▶ <https://googleprojectzero.blogspot.com/2018/01/reading-privileged-memory-with-side.html>
- ▶ <https://googleprojectzero.blogspot.com/2015/03/exploiting-dram-rowhammer-bug-to-gain.html>
- ▶ “Security analysis of AMD Predictive Store Forwarding”, AMD (may-2021)
<https://www.amd.com/system/files/documents/security-analysis-predictive-store-forwarding.pdf>
- ▶ <https://rambleed.com/>
- ▶ <https://m1racles.com/>
- ▶ <https://www.usenix.org/system/files/sec21-paccagnella.pdf>
- ▶ MIRAGE:
<https://www.usenix.org/conference/usenixsecurity21/presentation/saileshwar>
- ▶ <https://www.vusec.net/projects/bhi-spectre-bhb/>
- ▶ <https://www.prefetchers.info/>
- ▶ <https://pacmanattack.com/>
- ▶ <https://www.hertzbleed.com/>
- ▶ <https://comsec.ethz.ch/research/microarch/retbleed/>
- ▶ <https://aepicleak.com/>
- ▶ <https://blog.trailofbits.com/2024/01/16/leftoverlocals-listening-to-llm-responses-through-leaked-gpu-local-memory/>
- ▶ <https://lock.cmpxchg8b.com/zenbleed.html>
- ▶ <https://downfall.page/>
- ▶ <https://www.tomshardware.com/tech-industry/manufacturing/researchers-reveal-chips-that-commit-circuit-suicide-self-destruction-and-counterfeit-protection-in-one>
- ▶ <https://ghostwriteattack.com/>
- ▶ <https://lock.cmpxchg8b.com/reptar.html>
- ▶ <https://badram.eu/>
- ▶ <https://indirector.cpusec.org/>
- ▶ <https://bughunters.google.com/blog/5424842357473280/zen-and-the-art-of-microcode-hacking>
- ▶ <https://comsec.ethz.ch/research/microarch/branch-privilege-injection/>
- ▶ <https://www.vusec.net/projects/training-solo/>
- ▶ <https://predictors.fail/>
- ▶ <https://www.gpuhammer.com/>
- ▶ <https://www.vusec.net/projects/trrespass/>
- ▶ <https://comsec.ethz.ch/research/dram/phoenix/>

Perspectivas a futuro

- ▶ **¡Una época excitante en Arquitectura de Computadoras!**
 - ▶ La desaceleración de la Ley de Moore impulsa nuevas ideas y arquitecturas (DSA) para continuar mejorando la performance.
 - ▶ Inteligencia Artificial es un gran impulsor de nuevas arquitecturas (no Von Neumann).
 - ▶ El chip más grande de la historia demostró ser un éxito.
 - ▶ La seguridad y la flexibilidad deben ser consideradas en el diseño como nunca antes.
 - ▶ *¿Tendremos una Energy Wall? ¿Habrá una nueva Global Scale Computing?*
 - ▶ Un nuevo ISA en 30 años busca hacerles competencia a los dos gigantes que se reparten el mercado. Y tiene con qué.
 - ▶ El ISA de ARM está entrando en nuevas porciones de mercado (Apple, top500, Nvidia), algo que no había ocurrido antes.
 - ▶ *¿Habrá empezado una época de transición, y dentro de 10 años hablaremos sobre la arquitectura x86 como algo del pasado?*