Spectre: Vulnerabilidad a nivel de hardware

José Eduardo Guerra Sosa

***Abstract*—Spectre is a vulnerability, that together with Meltdown, was made public at the beginning of 2018 , this vulnerability is special since it affects the computer hardware instead of the software. This causes consequences that affect users in new or uncommon ways.**

As with any malware, it is very important to know what damage can cause, how to protect ourselves from this, taking preventive measures.

***Index Terms*—hardware, vulnerabilidad**

I. Introducción

Normalmente cuando se intenta crear un malware se busca encontrar fallos en el diseño del código de algún programa o sistema para así explotar estos errores. En este caso el malware es específico para ese programa o sistema operativo, motivo por el cual la mayoría de malware afecta a Windows, ya que es el sistema que más alcance permite. Pero ¿Qué pasa cuando se encuentra un fallo a nivel de hardware?, esto es lo que a pasado con Spectre, una de las dos vulnerabilidades encontradas y hechas públicas a principios de 2018, que son fallos que se encuentran en el diseño de unas características que existen en los CPUs modernos.

El objetivo de esta investigación es dar a conocer que consecuencias tiene este tipo de vulnerabilidades, y así lograr que los usuarios tomen conciencia de los peligros que existen y puedan tomar medidas preventivas para protegerse de malware que se aprovecha de esta vulnerabilidad.

Para esto se explicará a grandes rasgos cómo funciona esta vulnerabilidad para así comprender el problema que se está enfrentando, y con esto obtener conclusiones de la forma en la que personas malintencionadas pueden hacer uso indebido de este fallo, además saber qué medidas se pueden tomar para prevenir este tipo de ataques.

II. ¿Cómo el hardware de una computadora ejecuta los programas?

Para entender cómo funcionan estas vulnerabilidades, hay que saber cómo el hardware de la computadora ejecuta un programa. Un programa de computadora es un conjunto de instrucciones que la CPU tiene que ejecutar para su funcionamiento.

A. Proceso

Se puede definir un proceso, como un programa en ejecución [1]. El sistema operativo es el encargado de gestionar estos procesos.

José Eduardo Guerra Sosa is with Ingeniería en Ciencias y Sistemas, Centro Universitario de Oriente, USAC, e-mail: josguerra99@gmail.com

B. Memoria de acceso aleatorio (RAM)

Se denomina de acceso aleatorio porque puede leer y escribir en cualquier posición con un tiempo de espera igual. Se utiliza como la memoria principal de una computadora, en esta se cargan las instrucciones que necesitaran ser procesadas para la ejecución de un programa [2] , es decir los programas deben de estar en RAM para poder ser ejecutados.

Está compuesta por una gran matriz de bytes, cada uno con su propia dirección [3] , estos bytes pueden representar instrucciones que se deben realizar para continuar la ejecución de un programa, o información que un programa necesite por ejemplo valores de variables e incluso contraseñas.

Para proteger esta información a cada proceso se le asigna un espacio de memoria separado, este espacio de memoria tiene una base que almacena la dirección de memoria más pequeña a la que puede acceder y un límite que indica el tamaño que tiene el rango de direcciones. Además de separar los procesos, la memoria principal se divide en dos secciones, la de usuario que es el espacio que se le asigna a los procesos y el de núcleo o kernel qué es el del sistema operativo, éste tiene acceso a toda la memoria principal y contiene instrucciones privilegiadas. El encargado de revisar que se está accediendo a un espacio de memoria válido es al CPU, si un proceso intenta acceder a una dirección de memoria que no le pertenece, se producirá una interrupción haciendo que el sistema operativo de un error fatal [3].

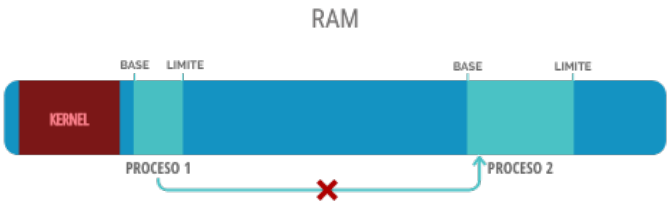

Fig. 1. Ejemplificación de procesos en memoria principal con su respectivo espacio de direcciones, el intento de un proceso de leer información que no le pertenece producirá un error.

C. Unidad central de procesamiento (CPU)

La unidad central de procesamiento o microprocesador realiza operaciones con los datos que se encuentran en memoria. La CPU estará ejecutando instrucciones de la siguiente manera: obtener la primera instrucción de memoria, decodificarla para determinar su tipo y operandos, ejecutarla y después obtener decodificar y ejecutar las instrucciones subsiguientes [4].

Por lo que podemos decir de forma muy simplificada, que las acciones que hace una CPU para ejecutar una instrucción son:

1) Obtener la instrucción
2) Decodificarla
3) Ejecutarla

Un procesador moderno es capaz de ejecutar miles de millones de instrucciones por segundo, este valor viene dado por factores como la frecuencia del microprocesador, actualmente está viene dada en Ghz y esta indica cuantos ciclos de reloj el procesador da cada segundo, 1 Ghz equivale a 109 ciclos por segundo, la frecuencia de un microprocesador moderno se encuentra comúnmente en el rango de 3 a 4 Ghz . Agregando a esto se tiene que tomar en cuenta el número de instrucciones por ciclo (IPC) [5] .

Un problema con esto, es que cuando se compara esta velocidad con la velocidad de la memoria principal (que en las más modernas oscila entre 2133 Mhz hasta los 4000 Mhz [6]) se puede notar que es demasiado lenta para la CPU, lo que provoca que no se aproveche toda la potencia que tiene, estaría más tiempo esperando por los datos de la memoria, que ejecutando las instrucciones, así que para resolver esto se creo una memoria especial llamada memoria caché.

1) Caché : Las memorias cachés son pequeñas (poca capacidad) y de alta velocidad, son utilizadas por los microprocesadores modernos; estas contienen información que se cree llegará a ser utilizada por el microprocesador [7].

La memoria caché contiene en cualquier momento una copia de una porción de la memoria principal [8].

Cuando la CPU necesita acceder a una palabra en la memoria principal, primero revisa la caché, si está allí accede a la caché y copia la palabra en caso contrario accede a la memoria y copia un bloque comenzado por la palabra deseada, y reemplaza el contenido en el que está en caché [8].

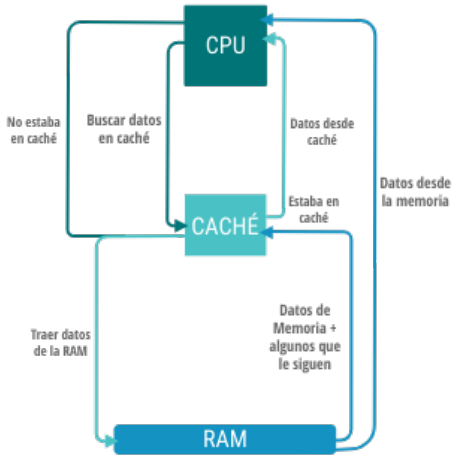

Fig. 2. Diagrama simplificado de los pasos que se realizan cuando el CPU necesita obtener datos que residen en memoria.

Las memorias cachés deben ser completamente transparentes al resto del sistema [9] , lo que significa que los programas no pueden acceder ni modificar directamente los datos que residen en ellas.

2) Segmentación (pipelining): Es otra forma de aumentar la velocidad de los microprocesadores. Esta es una técnica que permite traslapar la ejecución múltiples instrucciones [10]. Para esto se divide la ejecución de las instrucciones en las etapas mencionadas anteriormente, y de esta forma mientras el procesador realiza la etapa de una instrucción, puede estar realizando una etapa diferente de otra instrucción.

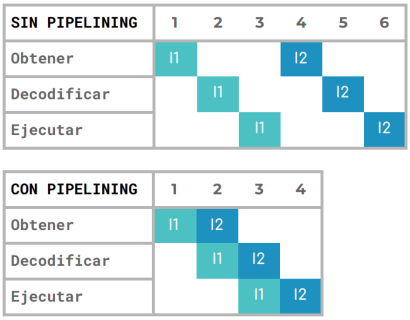

Fig. 3. Compartiva de la ejecución de dos instrucciones, una sin segmentación y otra con segmentación, al utilizar segmentación podemos ver que nos ahorramos dos "pasos"

3) Predictor de saltos (branch predictor): Un problema con la segmentación es que si una instrucción necesita el resultado de la otra, por ejemplo cuando existe una condición sí (if), acá necesitamos saber si el resultado de la condición es verdadero o falso antes de ejecutar el código que se encuentra dentro de ella, en este caso las instrucciones se ejecutarán en forma secuencial, no haciendo uso de la capacidad de ejecutar varias etapas a la vez del microprocesador [11].

Para intentar resolver este problema los microprocesadores tienen un circuito llamado predictor de saltos, y lo que este hará será ejecutar la siguiente instrucción de manera especulativa, para explicar qué es esto se usará el ejemplo de la condición if: Si es muy probable que el resultado de esta condición sea verdadero (hay métodos para intentar determinar esto usando ejecuciones anteriores del programa), entonces antes de terminar de comprobar que sea en realidad verdadero se empezará a ejecutar las instrucciones que se encuentren dentro de la condición sin hacer cambios permanentes (por ejemplo escribir a memoria principal). Si al final esté era verdadero entonces se guarda todo lo que las instrucciones anteriores realizaron, en caso contrario se descartan las acciones hechas por las instrucciones que se ejecutaron especulativamente y se pasa a ejecutar las instrucciones correctas [11].

III. Spectre

Spectre y Meltdown son vulnerabilidades que se hicieron públicas a principios de 2018, estas vulnerabilidades se dan a nivel de hardware, concretamente afectan el microprocesador. Al explotar estas vulnerabilidades es posible obtener información que se encuentra en áreas de la memoria principal que normalmente serían inaccesibles. Meltdown permitiría leer desde la memoria del núcleo y Spectre leer un espacio de memoria que le pertenece a otro proceso [12].

Cómo se mencionaba el objetivo de esta investigación es dar a conocer el impacto que tienen las vulnerabilidades de hardware y que acciones preventivas realizar, así que para fines prácticos se hará un análisis exclusivamente de Spectre.

Spectre afecta a todos los procesadores que utilizan predicción de saltos [13] , es decir todos los procesadores modernos,

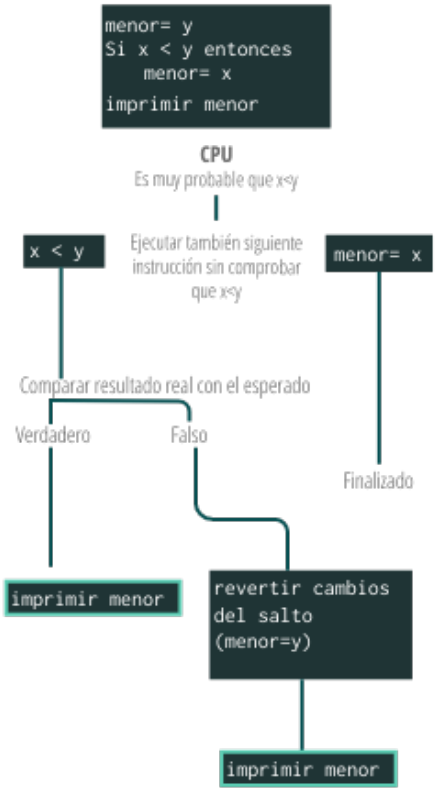

Fig. 4. Ejemplo de cómo se realiza una ejecución especulativa, y las posibles acciones que se deban tomar.

ya sean de Intel, AMD o incluso procesadores basados en ARM [12] , los cuales se encuentran en los dispositivos móviles.

Hay que tener en cuenta que la información a la que se puede acceder utilizando Spectre debe encontrarse en memoria principal, es decir que la máquina víctima debe ejecutar un programa, así que si un atacante quiere información específica, este deberá esperar a que el usuario ejecute un programa que contenga esa información, por ejemplo en el caso de una contraseña, debería esperarse a que el usuario inicie sesión con la contraseña.

Spectre es un ataque local, lo que significa que un malware que utilice Spectre debe estar ejecutándose en la máquina víctima. Eso no significa que el atacante necesita acceso físico, como indica Anders Fogh de Gdata:

> Ni Spectre ni Meltdown requieren acceso físico. Estos son puramente software, que solo utilizan cómo funciona el hardware
>
> *Anders Fogh*

Así que como con cualquier malware, este solo es software aprovechándose de alguna vulnerabilidad, en este caso de los microprocesadores. Y como con todo malware, se intenta hacer que el usuario víctima ejecute un programa que contenga el código malicioso; ya que Spectre se utilizará para robar información, también se necesita que el malware envie la información obtenida al atacante, todo esto se realiza sin que se tenga acceso físico a la máquina víctima.

Existen pruebas de concepto de Spectre utilizado desde JavaScript [14] , que se encuentra en casi todas las páginas de web, esto quiere decir que no es necesario instalar el malware en la máquina víctima, y este se ejecutaría cada vez que alguien visite la página web con código malicioso.

A. Análisis de prueba de concepto (PoC)

Para realizar un ataque con Spectre se deberían realizar los siguientes pasos [15] :

1) Entrenar al predictor de saltos para que ejecute especulativamente una instrucción.
2) Vaciar la caché.
3) Ejecutar una instrucción que no se encuentre en el espacio de memoria del proceso (ya que se entrenó el microprocesador entonces la ejecutara especulativamente).
4) Usar ese resultado para afectar la caché.

Para entender cómo funciona se analizará el código de una prueba de concepto realizada en JavaScript, PoC creada por alephsecurity [1] .

Algunas partes del código fueron modificadas para que sean más comprensibles.

1) Información acerca de la prueba de concepto: El objetivo de la prueba de concepto es intentar leer un número entero que se encuentra dentro de un arreglo. Hay que tener en cuenta que es una prueba de concepto y que se leerá de memoria que le pertenece al mismo proceso, pero la forma de leer estos datos será realizando los pasos de Spectre.

Esta prueba se realizó en Google Chrome versión 69.0.3497.100.

2) Análisis del código: El análisis del código se hará solamente en las partes más importantes, es decir, las que realizan los pasos mencionados anteriormente.

Para obtener el número se irá creando bit a bit, un entero tiene 32 bits.

a) Variables:

1) iterCnt es el número de veces que se intentará obtener cada bit, esto se hace por cualquier fallo que exista en la obtención del bit, así que al final se hará una proporción de los bit para saber cuál era el valor real.
2) datArr es un arreglo de tamaño dos, la primera posición se encuentra un 0 y la segunda posición tiene el número secreto que se intenta leer desde memoria.
3) bitIsZeroCntArr y bitIsOneCntArr son dos arreglos donde se almacenará que valor tuvo el bit en cada iteración, además se multiplica por dos el tamaño porque en cada iteración se realizarán dos comprobaciones, primero revisando si es un bit 1 y luego un 0, y la siguiente comprobación primero se revisa si es un bit 0 y luego un 1.
4) ratioArr contendrá la proporción de los bits, tamaño 32 porque necesitamos una proporción por cada bit del número, con esta información se puede reconstruir el número.
5) mainArrAccesor es un arreglo dividido en 4 partes, la primera es un arreglo para vaciar la caché, la segunda

[1] https://github.com/alephsecurity/spectreBrowserResearch Prueba de concepto de Spectre funcionando con JavaScript

un arreglo de exploración que es con el cual podremos saber si el bit es un 0 o un 1.

6) mainArrAccesor es un arreglo que se puede separar en 5 regiones:
   a) La primera parte servirá para vaciar la caché
   b) La segunda es un arreglo que tendrá valores que sean mayor a 1000, ya que se entrenará al predictor de saltos para que ejecute especulativamente si el valor es menor a 1000, en este caso al ser mayor, ejecutará especulativamente y luego se dará cuenta que no debió hacerlo.
   c) La tercera parte es un arreglo que se dividirá en dos partes, si el bit que se obtuvo era 0, se guardará en caché la primera parte, en caso de ser 1 guardará la segunda parte en caché.
   d) La tercera parte simplemente para entrenar al predictor de saltos, este es un arreglo parecido al anterior.
   e) Y en la última posición se encuentra con valor 0 que tambien servira para entrenar al predictor de saltos, en este caso a la condición ya que 0 es menor a 1000.

```
var iterCnt = 500;
var dataArr = new Int32Array(2);
dataArr[0] = 0;
dataArr[1] = numeroSecreto;
var bitIsZeroCntArr = new Int32Array(2 * iterCnt);
var bitIsOneCntArr = new Int32Array(2 * iterCnt);
var ratioArr = new Float32Array(32);
var mainArrAccessor = new Int32Array(flushArrSize+ cmpArrSize + probeArrSize * 2 +trainProbeArrSize * 2 + 1024);
```

Code Snippet 1. Inicialización de las variables

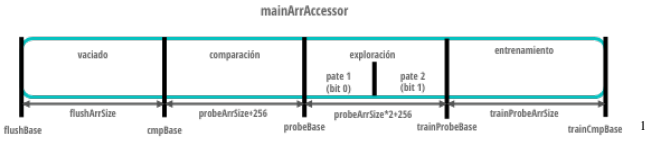

Fig. 5. Como se dividira el arreglo principal que se usará para leer los bits del número

b) Funciones:

_speculativeAccessFuncWithBitOffset: Esta función tiene una condición de comparación que servirá para entrenar al microprocesador haciendo que este ejecute especulativamente lo que hay dentro de la condición. La comparación será que un valor tiene que ser menor a 1000.

Fase de entrenamiento Acá el valor siempre será menor a 1000 (será 0 como se indicó anteriormente), por lo que el procesador empezará a pensar que este siempre será el caso y empezará a ejecutar especulativamente lo que se encuentra dentro de la comparación.

Tambien se leerá de dataArr[0] el cual tiene un valor de 0 como se indicó anteriormente.

Engañando al predictor de saltos Ahora sabemos que el procesador ejecutará el código que está dentro de la condición especulativamente, para asegurar que se termine de ejecutar este código antes de el microprocesador se dé cuenta que no debió ejecutarlo y revierta los cambios, se vacía la caché, para que este tenga que ir a memoria por mainArrAccessor[cmpBase+probeOffset] realizar esta acción será lento.

Aunque termine de ejecutar todo especulativamente, sabemos que se revertirán los cambios, pero para superar esto se hace lo siguiente.

Sabemos que se está ejecutando especulativamente el código que se encuentra dentro de la condición, así que primero leemos 1 de los 32 bit de dataArr[1] (bitOffset indica que bit se va a leer).

Si ese bit es 0, se leerá la primerá parte del arreglo de exploración, si es 1 se leerá la segunda parte, para esto se encuentra [(probeArrSize+512)*bit], al leer del arreglo de exploración, lo traerá de la memoria principal, y sabemos que cuando el procesador trae algo de memoria principal lo pondrá en caché.

El microprocesador revertirá los cambios, pero la primera o segunda parte del arreglo de exploración quedará en caché.

```
if (mainArrAccessor[cmpBase + probeOffset] < 1000) {
    var bit  = (dataArr[readIndex] >>> bitOffset) & 1;
    var indx = probeBase + probeOffset + ((probeArrSize + 512) * bit);
    t        = mainArrAccessor[indx] + t;
}
```

Code Snippet 2. Parte que se ejecutará especulativamente de la función _speculativeAccessFuncWithBitOffset

c) Pasos: Para leer un bit se realizan los siguientes pasos

Vaciar caché: Primero necesitamos vaciar la caché, esto servirá para que se logre terminar la ejecución en modo especulativo.

```
_cacheFlush(mainArrAccessor, flushBase, flushArrSize);
```

Code Snippet 3. Vaciar caché

Entrenar predictor de saltos: Ya teniendo la caché vacía entonces hay que entrenar al predictor de saltos. Para esto se llama a la función _speculativeAccessFuncWithBitOffset varias veces, con los datos de entrenamiento.

```
for (var i1 = 0; i1 < 5; i1++) {
  _speculativeAccessFuncWithBitOffset(mainArrAccessor, trainProbeBase, 0, trainProbeArrSize, trainCmpBase, dataArr, 0, 0, useSpeculative);
}
```

Code Snippet 4. Entrenar al microprocesador para que utilice ejecución especulativa

Engañar al predictor de saltos: Luego de que se entrenará al predictor de saltos para que ejecute especulativamente

```
_speculativeAccessFuncWithBitOffset(
    mainArrAccessor, probeBase, prOffset,
     probeArrSize, cmpBase, dataArr, 1,
    bit, useSpeculative);
```

Code Snippet 5. Leer el número con ejecución especulativa

Ahora sabemos que dependiendo del bit que se haya leído hay una parte del arreglo de exploración dentro de la caché, pero no podemos leer la caché directamente, para esto se realiza otro truco.

Suponiendo que el bit haya sido un 1, entonces si leemos la segunda parte del arreglo de exploración, el procesador lo leerá de la memoria caché y no de la memoria principal, por lo que debería ser muy rápido, así que la siguiente parte se basa en temporizadores muy precisos, en JavaScript existe performance.now(), que da el tiempo con precisión de 5 microsegundos [16] .

Así que se hará lo siguiente:

1) Guardar el tiempo actual en una variable, llamado lastTick
2) Leer de la segunda parte del arreglo de exploración
3) Si lastTick sigue siendo igual a performance.now() significa que no han pasado más de 5 microsegundos, por lo que fue demasiado rápido y es casi seguro que el dato se trajo de caché. En este caso tomamos el bit como un 1.
4) Luego de esto se repite lo mismo pero con la primera parte del arreglo de exploración para un dato 0.

```
idx = probeBase + probeArrSize + 512;
lastTick = performance.now();
while (idx <
   probeBase + 2 * probeArrSize) {
    idx = mainArrAccessor[idx];
}
while (lastTick == (performance.now())) {
    bitIsOneCnt++;
}
```

Code Snippet 6. Comprobar si dato se encontraba en caché para saber si era un 1 o un 0

Todos los pasos anteriores (desde borrar la caché) se repiten pero esta vez se comprueba primero si el bit es un 0 (leer primera parte del arreglo de exploración) y luego el 1.

Eso serían los pasos para obtener un bit una sola vez. Todos esos pasos se repiten por la cantidad que se puso en iterCnt, para luego obtener una proporción y poder estar más seguros de que era un 1 o un 0. Luego todo esto se repite por los 32 bits que tiene un entero.

```
for (bit = 31; bit >= 0; bit--) {
    for (j = 0; j < iterCnt; j++) {
       pasosPrimero1Luego0( );
        bitIsOneCntArr[2 * j] =
            bitIsOneCnt;
        bitIsZeroCntArr[2 * j] =
            bitIsZeroCnt;
        pasosPrimero0Luego1( );
        bitIsOneCntArr[2*j + 1] =
            bitIsOneCnt;
        bitIsZeroCntArr[2* j + 1] =
            bitIsZeroCnt;
    }
    ratioArr[bit]=(promedio(
       bitIsZeroCntArr) /promedio(
       bitIsOneCntArr));
}
```

Code Snippet 7. Obtener todos los bits

Sólo resta convertir las proporciones a un número entero, para esto se llama a la función bitsToInt que recibe como parámetro el arreglo de proporciones.

Los investigadores se dieron cuenta que dependiendo del navegador una proporción mayor que uno no significaba que el bit sea un 0, en el caso de Chrome, una proporción mayor a 0.95 significaba que el bit era un 0.

```
function bitsToInt(bits){
  var val=0;
  for (bit = 31; bit >= 0; bit--) {
    val= (bits[bit]>0.95)?(val <<
       1) | 0:(val << 1) | 1;
  }
  return val;
}
```

Code Snippet 8. Función que convierte el arreglo de proporciones a un número entero

Para esta prueba el resultado de la función bitsToInt se pondrá en un div, además de unos logs en consola para entender cómo se van obteniendo las proporciones de los bits y cómo van formando el número. Figures 6, 7 and 8

B. Riesgos que se presentan

Para entender mejor porque este tipo de vulnerabilidades difieren de las de software, hay que saber que efectos pueden llegar a tener:

1) Al afectar al hardware, en el caso de Spectre a la mayoría de microprocesadores, todos los dispositivos que los usen son vulnerables independientemente del sistema operativo que se utilice.
2) Para resolver completamente el error se necesita reemplazar el hardware; aunque se puede mitigar el efecto con actualizaciones del sistema operativo.
3) Pueden afectar a cualquier programa que se encuentre instalado en el dispositivo.

Una de las amenazas más grandes que presenta el uso de Spectre es el robo de información, ya que se puede leer memoria de otros procesos que contengan datos como: contraseñas, cuentas bancarias, mensajes privados, entre otros. Esto puede resultar daños para el usuario como pérdida económica o robo de identidad [17] .

Otra amenaza que se da con el poder acceder a datos que se encuentran en memoria, es que esos datos pueden ser utilizados por un hacker en una fase de reconocimiento,

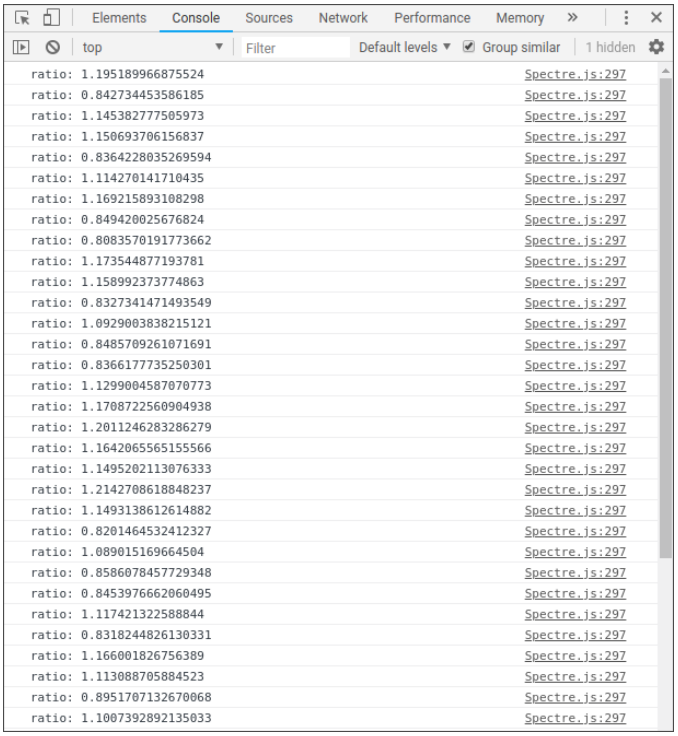

Fig. 6. Las proporciones que va obteniendo de cada bit (>0.95 es un 0)

Fig. 7. Al ir convirtiendo las proporciones en bits se va generando el número

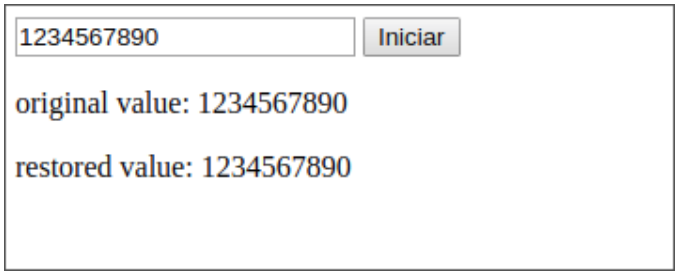

Fig. 8. Resultado final, se puede comprobar que el valor original y el obtenido son el mismo

en la cual intentará obtener información útil para un futuro ataque [18] .

El alcance que tiene Spectre es muy vasto ya que al afectar a casi todos los microprocesadores modernos, este iría desde computadoras y móviles personales, hasta servicios de computación en la nube, servidores entre otros [12] .

Un impacto muy grande puede ser en los servicios en la nube, ya que estos son multitenant es decir que todos utilizan los mismos recursos, aunque los recursos de cada cliente son visibles sólo para él [19] , alguien que utilice Spectre podría ver información que le pertence a alguien más.

C. *Acciones que se han tomado*

Antes de hacer públicas las vulnerabilidades, las grandes empresas fueron notificados, y estuvieron trabajando en soluciones para Meltdown y Spectre.

Los parches contra Meltdown son más efectivos por ejemplo del lado del núcleo de Linux se añadió la característica KPTI (aislamiento de tablas de páginas del núcleo); Windows y OS X también tienen actualizaciones para mitigar Meltdown. [12]

En cambio para de Spectre, las actualizaciones logran hacer más dificil el trabajo de explotar la vulnerabilidad, pero no significa que el problema este resuelto [12] . Por ejemplo para el compilador MSVC se añadió la opción /Qspectre, que inserta código para mitigar Spectre, este código inserta instrucciones que actúan como una barrera de ejecución de código especulativo [20] .

Uno de los efectos secundarios de estas actualizaciones es que afectan el rendimiento de los programas o sistema operativo, el impacto siendo entre el 1 y el 8% [21].

D. *Consecuencias*

Expertos han identificado 130 malware que intentan aprovecharse de Spectre y Meltdown, aunque estos se encontraban en una fase previa o de pruebas [22].

Además del malware que puede ser utilizado, todas los parches que han salido para combatir Spectre y Meltdown han hecho que el rendimiento de los sistemas disminuya, afectando a los usuarios e incluso proveedores como los que ofrecen servicios en la nube.

Con el surgimiento de estas dos vulnerabilidades se han encontrado aún más fallos, basados en estos mismos [23].

E. Recomendaciones

Revisar si el sistema es vulnerable, para eso existen varias herramientas en línea, una de ellas es spectre-meltdown-checker [2]

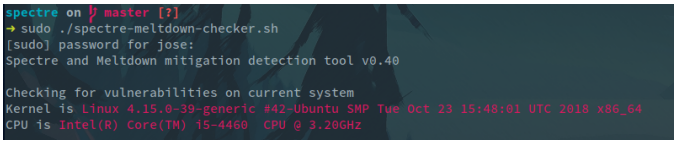

Fig. 9. spectre-meltdown-cheker obtendra el núcleo y el hardware del sistema

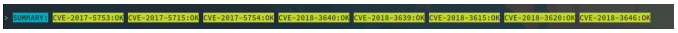

Fig. 10. Al final mostrará un resumen de las pruebas que realizo, si muestra OK es que el sistema tiene instaladas todas las actualizaciones que mitigan las vulnerabilidades.

No queda de más actualizar el sistema operativo a la última versión y también aplicaciones como navegadores web.

También hay que tener en cuenta que esto será malware que se aprovecha de las vulnerabilidades de hardware, asi que tambien hay que protegerse de este:

1) Navegando únicamente en sitios seguros.
2) No ejecutando programas de fuentes desconocidas.
3) Teniendo un antivirus actualizado.

Si es posible, renovar el hardware actual por nuevas versiones que ya no sean vulnerables.

IV. Conclusión

A medida que se van descubriendo vulnerabilidades que pueden llegar a afectar a los usuarios finales, el software a su vez es más seguro, se siguen creando cada vez mejores prácticas de programación enfocadas a seguridad, que logran hacer que los programas sean menos susceptibles a ataques. Estas prácticas las deben obligatorias para programas donde se guarde información sensible de los usuarios, por ejemplo contraseñas y cuentas bancarias.

Pero incluso con las mejores prácticas de programación, cuando existe una vulnerabilidad en el hardware, cualquier programa puede resultar afectado. Y cuando algo como esto sucede, casi ninguna aplicación puede hacer algo al respecto, ya que estas no tienen un control directo sobre el hardware y si pareciera que lo están utilizando, por ejemplo lecturas al disco o asignación de memoria principal, estas en realidad son herramientas proporcionadas por el sistema operativo, qué es el único que tiene comunicación directa con la mayoría de hardware de la computadora. Por eso es que algunas de las amenazas que se pueden dar a través hardware pueden ser mitigadas por actualizaciones en el sistema operativo.

Aunque existan actualizaciones en el sistema operativo para combatir este tipo de amenazas, éstas siempre traen efectos secundarios como por ejemplo hacer más lento el sistema o una mayor infrautilización de los recursos, y aún con esto, el verdadero fallo no estaría arreglado, ya que para eso se necesitaría renovar el hardware de la computadora por uno rediseñado para no tener las vulnerabilidades que provocaron el cambio inicialmente.

Cuando se piensa en un rediseño los fabricantes de hardware tienen un reto muy grande, no se puede simplemente quitar la característica que lo hace vulnerable, tomando el caso de Spectre, no pueden simplemente quitar la predicción de saltos, gracias a esta característica los microprocesadores tienen la velocidad por la que son reconocidos hoy en día, quitar esta característica sería dar muchos pasos hacia atrás. Así que necesita ser un rediseño de la forma en la que funciona la característica; y aunque encuentren una solución viable, hay que tener en cuenta que los fabricantes ya tenían planeado el diseño de sus futuros productos, lo que implica que estos deberán ser modificados por el nuevo diseño.

Y por eso es que una vulnerabilidad de hardware puede llegar a ser una mayor amenaza que una de software, el alcance que tiene es mas grande, solucionarla es más difícil y costosa, no solo para los fabricantes, si no que los usuarios también deberán comprar el nuevo hardware para estar completamente seguros. Todo esto no significa que se pueda evitar tener vulnerabilidades, así como pasa con el software sabemos que siempre habrán personas intentando encontrar agujeros en el código que les permitan realizar acciones que no se encontraban a propósito, también habrán personas que busquen lo mismo en el hardware, y es cierto, es posible que logren encontrar alguna vulnerabilidad cada cierto tiempo, pues los desarrolladores y fabricantes no pueden pensar en todos los fallos que existen en el diseño ya sea de código o de hardware; pero a su vez, cuando esto pasa, el software y hardware también se volverán más seguros.

[2]https://github.com/speed47/spectre-meltdown-checker Script de shell para comprobar si el sistema es vulnerable a Meltdown y Spectre

References

[1] J. C. Pérez, P. de Miguel Anasagasti, and F. G. Carballeira, "SISTEMAS OPERATIVOS Una visión aplicada," C. F. Madrid, Ed. MC Graw Hill, 8 2001, pp. 77–80.
[2] Wikipedia, "Memoria de acceso aleatorio." [Online]. Available: https://es.wikipedia.org/wiki/Memoria_de_acceso_aleatorio
[3] A. Silberscahtz, P. B. Galvin, and G. Gagne, "Fundamentos de sistemas operativos." McGraw-Hill, 2006, pp. 243–247.
[4] A. S. Tanenbaum, "Sistemas operativos modernos." México: Pearson Educación, 2009, pp. 19–23.
[5] C. P. U. World, "CPU Frequency." [Online]. Available: http://www.cpu-world.com/Glossary/C/CPU_Frequency.html
[6] J. Gomzar, "Escalado de la memoria DDR4 de 2133 hasta los 4000 MHz," 04 2016. [Online]. Available: https://www.profesionalreview.com/2016/05/23/escalado-de-la-memoria-ddr4-de-2133-hasta-los-4000-mhz/
[7] A. J. Smith, "Cache Memories ," *ACM Computing Surveys (CSUR)*, vol. 14, pp. 473–530, 1982. [Online]. Available: 10.1145/356887.356892
[8] B. A. Forouzan, "Introducción a la ciencia de la computación: De la manipulación de datos a la teoría de computación." Cengage Learning Latin America, 06 2003, pp. 75–76.
[9] U. Drepper, "What Every Programmer Should Know About Memory," 11 2007.
[10] "Pipelining." [Online]. Available: http://web.cs.iastate.edu/~prabhu/Tutorial/PIPELINE/pipe_title.html
[11] A. Fog, "The microarchitecture of Intel, AMD and VIA CPUs ," 2018. [Online]. Available: https://www.agner.org/optimize/microarchitecture.pdf
[12] G. U. of Technology, "Meltdown and Spectre," 2018. [Online]. Available: https://meltdownattack.com/
[13] Wikipedia, "Wikipedia." [Online]. Available: https://es.wikipedia.org/wiki/Spectre_(vulnerabilidad)

[14] K. Paul, H. Jann, F. Anders, D. Genkin, , G. Daniel, H. Werner, H. Mike, L. Moritz, M. Stefan, P. Thomas, S. Michael, and Y. Yuval, "Spectre Attacks: Exploiting Speculative Execution," in *40th IEEE Symposium on Security and Privacy (S&P'19)*, 2018.
[15] M. Godbolt, "Meltdown And Spectre," 01 2018. [Online]. Available: https://www.youtube.com/watch?v=IPhvL3A-e6E
[16] developer.mozilla.org, "performance.now()." [Online]. Available: https://developer.mozilla.org/es/docs/Web/API/Performance/now
[17] osi.es, "El robo de identidad y sus consecuencias," 06 2011. [Online]. Available: https://www.osi.es/es/actualidad/blog/2011/06/02/el-robo-de-identidad-y-sus-consecuencias
[18] ehack, "Las fases del Hacking Ético," 05 2017. [Online]. Available: http://ehack.info/las-fases-del-hacking-etico/
[19] M. Rouse, "multi-tenant cloud." [Online]. Available: https://searchcloudcomputing.techtarget.com/definition/multi-tenant-cloud
[20] Microsoft, "/Qspectre," 11 2018. [Online]. Available: https://docs.microsoft.com/en-us/cpp/build/reference/qspectre?view=vs-2017
[21] redhat, "Speculative Execution Exploit Performance Impacts - Describing the performance impacts to security patches for CVE-2017-5754 CVE-2017-5753 and CVE-2017-5715," 03 2018. [Online]. Available: https://access.redhat.com/articles/3307751
[22] I. Ros, "Descubren malware que intenta aprovechar Spectre y Meltdown," 02 2018. [Online]. Available: https://www.muycomputer.com/2018/02/01/malware-aprovechar-spectre-meltdown/
[23] Jorgé, "Four new critical Spectre CPU vulnerabilities uncovered," 05 2018. [Online]. Available: https://react-etc.net/entry/four-new-critical-spectre-cpu-vulnerabilities-uncovered