HW4

學號:0716049
姓名:詹凱傑

Speculative Execution:

CPU 遇到 branch，或是一些情況時，可能會先預測接下來會執行哪一個指令，並先執行，如果預測正確則可以減少執行時間。現在的 CPU，執行指令時可能會 Out of Order，Meltdown 就是利用這個特性來攻擊，當 CPU 發生 exception 時，CPU 會預測並先執行和發生 exception 指令不相關的指令。

Flush+Reload:

透過 CPU 存取資料的時間差，可以知道資料放的位置是在 Cache 裡還是在 Memory 裡。透過這個特性，可以設計演算法來猜測某個記憶體的位置裡放的是什麼資料。

Meltdown exploits:

Meltdown 的攻擊流程簡略說明如下，由這次作業的例子為例，當 CPU 執行到會發生 exception 的指令後，會因為 Speclative Execution 先做下一行指令，並將這個指令的執行結果放到 Cache 中，由於資料從 Memory 到 Cache 時，不會做合法性的檢查，要到 register 時會做檢查，所以這個資料就會被存到 Cache 中。雖然真的到各個指令時，會因為存取不合法的位址讓這個指令失敗，但資料已經 load 到 Cache 中。這時候再透過 Flush+Reload 的手法來建造 Cache 中的資料，就可以讀到不屬於這個 process 的記憶體位置的資料。(在 Task 1 會用 spec 的 Code 說明)

Task 1:

```
1. int data = 34;
2. char kernel_data = *(char*)kernel_addr; /*exception occurred because user is not allowed to read kernel data*/
3. probe_array[data*4096+DELTA] +=1;
```

在第二行時，會觸發 exception，然後 CPU 會認為第三行和第二行沒關係，就會先執行第三行，這時 probe_array[34*4096+DELTA]會被 load 到 Cache。所以當我們去計算每個 probe_array 的 access time 會發現 probe_array[34*4096+DELTA]這段位置的 access time 特別快，可能在 cache 中，進而推論出 data 是 34。

實際操作(透過助教提供的 VM):

1. 先將 SecretModule.ko 載入 kernel，並查看記憶體位置:

```
user@ubuntu:~/Documents/os_hw4$ sudo cat /proc/kallsyms | grep secret
ffffffff8d74bd40 t move_master_key_secret
ffffffff8d8bf490 T crypto_stats_kpp_compute_shared_secret
ffffffff8d8bf510 T crypto_stats_kpp_set_secret
ffffffff8d8c5ac0 t dh_set_secret
ffffffff8de39810 t addrconf_sysctl_stable_secret
ffffffff8e888c40 r __ksymtab_crypto_stats_kpp_compute_shared_secret
ffffffff8e888c58 r __ksymtab_crypto_stats_kpp_set_secret
ffffffff8e8a48df r __kstrtab_crypto_stats_kpp_compute_shared_secret
ffffffff8e8a492b r __kstrtab_crypto_stats_kpp_set_secret
ffffffff8ec4e5c0 d ts_secret
ffffffff8ec4e5d0 d net_secret
ffffffff8ec50e20 d inet_ehash_secret.75043
ffffffff8ec50ec4 d udp_ehash_secret.79817
ffffffff8ec51120 d syncookie_secret
ffffffff8ec51550 d udp_ipv6_hash_secret.78523
ffffffff8ec51554 d udp6_ehash_secret.78522
ffffffff8ec515e0 d syncookie6_secret
ffffffff8ec52650 d ipv6_hash_secret.72752
ffffffff8ec52654 d inet6_ehash_secret.72751
ffffffff8f170c20 b ip4_frags_secret_interval_unused
ffffffff8f176a08 b ip6_frags_secret_interval_unused
ffffffffc0568380 b secret_buffer        [SecretModule]
ffffffffc0567168 r secret       [SecretModule]
```

2. 執行 ./toy.o ffffffffc0567169

結果:

```
ffffffffc0567168 r secret       [SecretModule]
user@ubuntu:~/Documents/os_hw4$ ./toy.o ffffffffc0567168
time of accessing elements in probe_array[0*4096]: 164
time of accessing elements in probe_array[1*4096]: 792
time of accessing elements in probe_array[2*4096]: 1009
time of accessing elements in probe_array[3*4096]: 577
time of accessing elements in probe_array[4*4096]: 676
time of accessing elements in probe_array[5*4096]: 1043
time of accessing elements in probe_array[6*4096]: 807
time of accessing elements in probe_array[7*4096]: 646
time of accessing elements in probe_array[8*4096]: 603
time of accessing elements in probe_array[9*4096]: 575
time of accessing elements in probe_array[10*4096]: 593
time of accessing elements in probe_array[11*4096]: 1501
time of accessing elements in probe_array[12*4096]: 539
time of accessing elements in probe_array[13*4096]: 595
time of accessing elements in probe_array[14*4096]: 456
time of accessing elements in probe_array[15*4096]: 468
time of accessing elements in probe_array[16*4096]: 603
time of accessing elements in probe_array[17*4096]: 585
time of accessing elements in probe_array[18*4096]: 478
time of accessing elements in probe_array[19*4096]: 1269
```

由此可以看出，像是 probe_array[0*4096]的 access time 就很短，所以有可能這段資料放在 cache 中，而像 probe_array[11*4096]的 access time 就比較長，所以可能是在 memory 中。

此外，從這邊我發現很多 probe_array 的 access time 大概落在 400 到 600 之間，所以我 Task 2 的 time_threshold 就先設定 500。

Task2:

透過作業中提供的 Meltdown_attack 來讀取剛剛載入 kernel 的 SecretModule.ko。

結果:

```
ffffffffc0567168 r secret       [SecretModule]
user@ubuntu:~/Documents/os_hw4$ ./Meltdown_attack ffffffffc0567168 7 500
The secret value at ffffffffc0567168 is 83 S 998/1000
The secret value at ffffffffc0567169 is 85 U 996/1000
The secret value at ffffffffc056716a is 67 C 1000/1000
The secret value at ffffffffc056716b is 67 C 999/1000
The secret value at ffffffffc056716c is 69 E 999/1000
The secret value at ffffffffc056716d is 101 e 999/1000
The secret value at ffffffffc056716e is 100 d 1000/1000
user@ubuntu:~/Documents/os_hw4$ ./Meltdown_attack ffffffffc0567168 7 300
The secret value at ffffffffc0567168 is 83 S 999/1000
The secret value at ffffffffc0567169 is 85 U 988/1000
The secret value at ffffffffc056716a is 67 C 989/1000
The secret value at ffffffffc056716b is 67 C 987/1000
The secret value at ffffffffc056716c is 69 E 973/1000
The secret value at ffffffffc056716d is 101 e 971/1000
The secret value at ffffffffc056716e is 100 d 986/1000
```

有此可以看出 ffffffffc0567168 之後 7 個 byte 放的資料是 SUCCEed。且用一般使用者的權限就可以看出這個結果，不用用到 super user。

Task3:

透過 spec 中的方法，來軟體的方式修補這個問題。

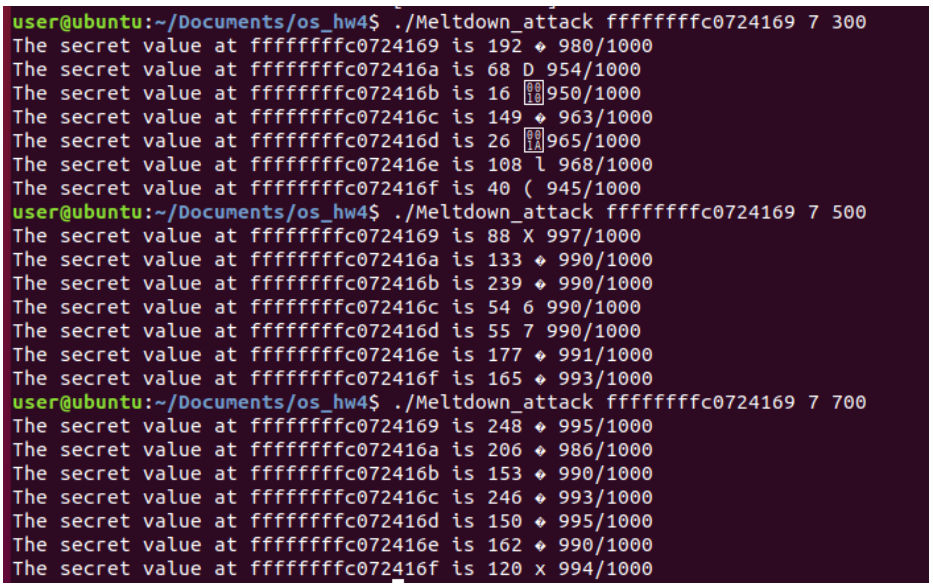
1. 修改 /etc/default/grub，並 update-grub 和 reboot

```
File  Edit  View  Search  Terminal  Help
# If you change this file, run 'update-grub' afterwards to update
# /boot/grub/grub.cfg.
# For full documentation of the options in this file, see:
#   info -f grub -n 'Simple configuration'

GRUB_DEFAULT=0
GRUB_TIMEOUT_STYLE=hidden
GRUB_TIMEOUT=0
GRUB_DISTRIBUTOR=`lsb_release -i -s 2> /dev/null || echo Debian`
GRUB_CMDLINE_LINUX_DEFAULT="quiet"
GRUB_CMDLINE_LINUX="find_preseed=/preseed.cfg auto noprompt priority=critical locale=en_US"

# Uncomment to enable BadRAM filtering, modify to suit your needs
# This works with Linux (no patch required) and with any kernel that obtains
```

2. 重新載入 SecretModule.ko

```
user@ubuntu:~/Documents/os_hw4$ sudo cat /proc/kallsyms | grep secret
ffffffffb3d4bd40 t move_master_key_secret
ffffffffb3ebf490 T crypto_stats_kpp_compute_shared_secret
ffffffffb3ebf510 T crypto_stats_kpp_set_secret
ffffffffb3ec5ac0 t dh_set_secret
ffffffffb4439810 t addrconf_sysctl_stable_secret
ffffffffb4e88c40 r __ksymtab_crypto_stats_kpp_compute_shared_secret
ffffffffb4e88c58 r __ksymtab_crypto_stats_kpp_set_secret
ffffffffb4ea48df r __kstrtab_crypto_stats_kpp_compute_shared_secret
ffffffffb4ea492b r __kstrtab_crypto_stats_kpp_set_secret
ffffffffb524e5c0 d ts_secret
ffffffffb524e5d0 d net_secret
ffffffffb5250e20 d inet_ehash_secret.75043
ffffffffb5250ec4 d udp_ehash_secret.79817
ffffffffb5251120 d syncookie_secret
ffffffffb5251550 d udp_ipv6_hash_secret.78523
ffffffffb5251554 d udp6_ehash_secret.78522
ffffffffb52515e0 d syncookie6_secret
ffffffffb5252650 d ipv6_hash_secret.72752
ffffffffb5252654 d inet6_ehash_secret.72751
ffffffffb5770c20 b ip4_frags_secret_interval_unused
ffffffffb5776a08 b ip6_frags_secret_interval_unused
ffffffffc0725380 b secret_buffer        [SecretModule]
ffffffffc0724168 r secret       [SecretModule]
```

3. 並透過 Task 2 的手法在試一次
   結果:



可以發現結果都變亂碼，沒辦法讀出 SUCCed。

kpti patches: 會將使用者的 process 使用的 page table 和 kernel 的 page table 分開。這樣 user 的 process 要讀取 Kernel 的資料時，就沒辦法映射到對的位置。

Conclusion:

我覺得這次的作業很有趣，之前就有聽教授說過 Meltdown，但我只知道這是 CPU 設計的漏洞，具體是怎麼攻擊的不太了解。不過在這次的作業後，讓我對 Meltdown 有更多的認識，且自己照著 spec 做一次後，並深入研究其中的原理後，我覺得這個攻擊非常厲害，特別是利用時間差這個資訊也可以當作攻擊的手段，讓我又多知道了一種攻擊方式。最後，我覺得在這次作業中我學到很多，特別是結合了計組的知識，讓我覺得我之前學的東西變得很實用。