HW4

學號:0716049

姓名:詹凱傑

Speculative Execution:

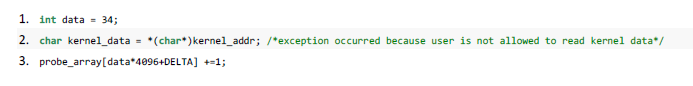
CPU遇到branch，或是一些情況時，可能會先預測接下來會執行哪一個指令，並先執行，如果預測正確則可以減少執行時間。現在的CPU，執行指令時可能會Out of Order，Meltdown 就是利用這個特性來攻擊，當CPU發生exception時，CPU會預測並先執行和發生exception指令不相關的指令。

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:

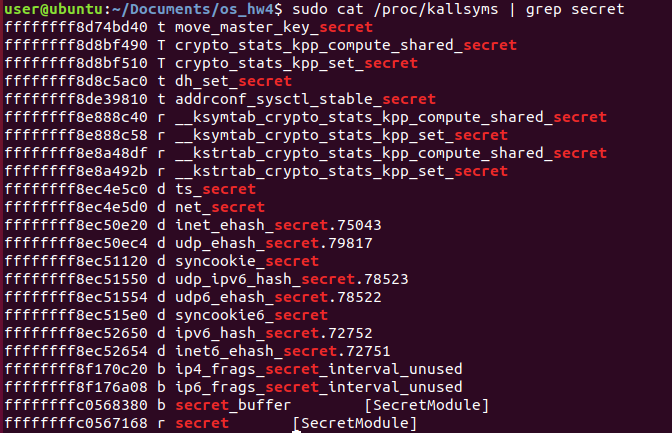
Meltdown的攻擊流程簡略說明如下，由這次作業的例子為例，當CPU執行到會發生exception的指令後，會因為Speclative Execution先做下一行指令，並將這個指令的執行結果放到Cache中，由於資料從Memory到Cache時，不會做合法性的檢查，要到register時會做檢查，所以這個資料就會被存到Cache中。雖然真的到各個指令時，會因為存取不合法的位址讓這個指令失敗，但資料已經load到Cache中。這時候再透過Flush+Reload的手法來建造Cache中的資料，就可以讀到不屬於這個process的記憶體位置的資料。(在Task 1會用spec的Code說明)

Task 1: 

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):

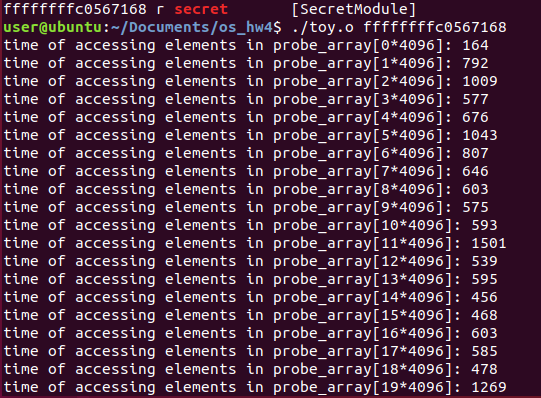
1. 先將SecretModule.ko 載入kernel，並查看記憶體位置:





1. 執行 ./toy.o ffffffffc0567169

結果:

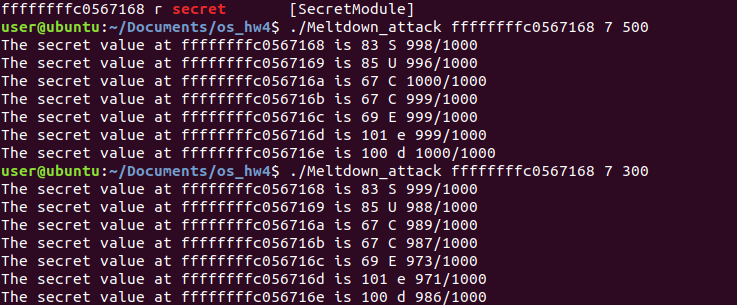


由此可以看出，像是probe\_array[0\*4096]的access time就很短，所以有可能這段資料放在cache中，而像probe\_array[11\*4096]的access time就比較長，所以可能是在memory中。

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。

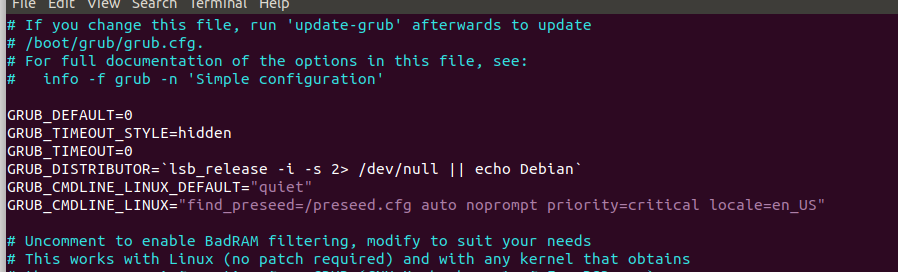
 結果:

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。

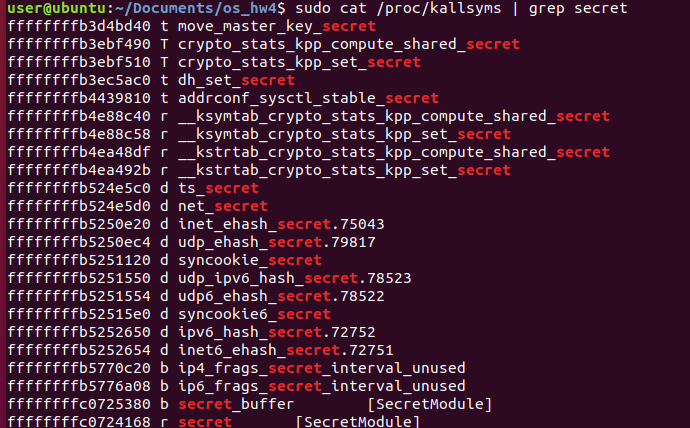
Task3:

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



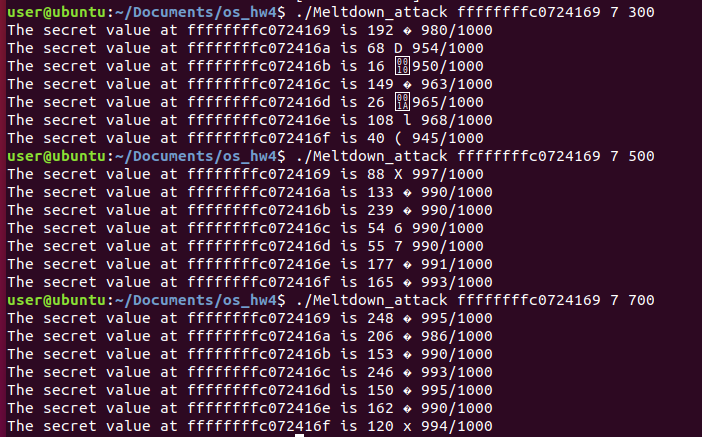
1. 重新載入SecretModule.ko





1. 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。