OS HW4:Meltdown

Meltdown: Reading Kernel Memory from user space (2018)

- Link

  https://www.usenix.org/system/files/conference/usenixsecurity18/sec18-lipp.pdf

  https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-5754

  The CPUs vulnerable to Meltdown

  https://www.techarp.com/guides/complete-meltdown-spectre-cpu-list/

Please make sure your CPU is vulnerable to Meltdown, or you may not run Meltdown successfully.

Introduction

- Memory isolation is commonly used in recent operating systems. It ensures that the user programs can not access kernel memory or other programs' memory, which makes running multiple processes simultaneously on one device possible.
- However, meltdown can break memory isolation based on poorly-designed digital system with the effects of Out-of-order execution and speculative execution on modern processors, so that a user process can randomly read kernel memory without privilege.

Secret data

The linux kernel module is provided, It stores secret_data in kernel space (SecretModule)

run

$ make

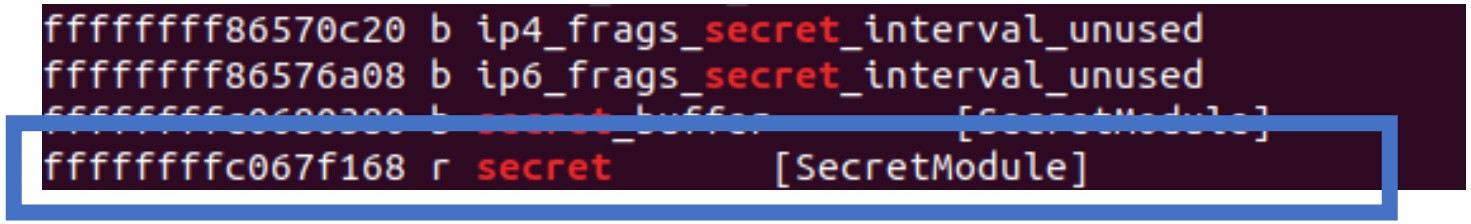
$ sudo insmod SecretModule.ko

to insert kernel module

To simplify the attack, we use sudo to directly get the secret data address

$ sudo cat /proc/kallsyms | grep secret

Secret data (cont.)



This is your target secret address in Task 2&3, I placed 7 bytes [S,U,C,C,E,e,d] in secret

Task 1: run toy.o

- Check the code below (part of toy.c)

```
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;
```

As a normal user, we are not allowed to read data from kernel memory, so Line 2 will trigger the exception, and the following codes are not supposed to be executed.

However, toy.o uses flush+reload to measure the access time of each page of probe_array, try running

$ ./toy.o [kernel_addr]

Kernel_addr here you can randomly pick a kernel address or just use the secret address you got from the last page.

What do you observe?

Explain the result with the provided code above.

Task 2: run Meltdown_attack

What if we replace data with kernel_data? (according to code on the previous page)

```
1. int data = 34;
2. char kernel_data = *(char*)kernel_addr; /*exception occurred*/
3. probe_array[kernel_data*4096+DELTA] +=1;
```

Try running

$ ./Meltdown_attack [secret_addr] [number of bytes to read] [time_threshold]

time_threshold: Observe the result you got from toy.o, set an appropriate time threshold that can distinguish between the access times from kernel and from cache.

針對讀取每個address的內容，由於亂序執行不一定每次都可以成功(competitive)，Meltdown_attack會跑1000次，取hit最多次的結果。

Could you leak secret (SUCCEed) placed in SecretModule with user privilege?

Show your result.

Task 3: Software Patch of Meltdown

- As we know, Ubuntu 18.04 has patched Meltdown already (software).
- Edit /etc/default/grub

  Delete nopti from GRUB_CMDLINE_LINUX_DEFAULT
- Run update-grub and reboot (you should insert SecretModule again after reboot)
- https://github.com/speed47/spectre-meltdown-checker

You can use this open source on github to check if you are vulnerable to Meltdown(CVE-2017-5754)

Run MeltdownAttack again to see if you can still read the secret.

Compare your result with Task 2 and explain how kpti patches this vulnerability.

Submission

- Simply describe how Meltdown exploits OOO execution/Speculative Execution/Flush+Reload to launch attacks
- Task 1 (what you observed and explain it)
- Task 2 (the result you get)
- Task 3 (compare the results before and after patch and explain pti mitigation for Ubuntu)
- What do you learn from this homework?/Conclusion
- Save it as HW4_[your Student ID].pdf and submit