Computer Security **Name:** Ameer Hamza
Assignment#3: Spectre Attack Lab

1. **Side Channel Attacks via CPU Caches**

   (a) **Task 1: Reading from Cache versus from Memory**: Check the 10 runs of the `Cache.c` program below. For most accesses, `array[3*4096]` and `array[7*4096]`, the times were less than 100 CPU cycles. However, 4th run has 117 and 123 cycles for both cache hits respectively. Also, 5th run is strange with `array[7*4096]` access going 562 CPU cycles. We can count this as an outlier and consider other values. Since the max CPU cycles for cache accesses were 123, we can safely consider **130** CPU cycles as threshold, as no cache hit is above 130, and all cache misses seem to be way above 130 cycles.

```
[03/16/23]seed@VM:~/.../Labsetup$ for i in {1..10}; do echo "accessing ${i}th time"; ./cache ; done
accessing 1th time
Access time for array[0*4096]: 2156 CPU cycles
Access time for array[1*4096]: 224 CPU cycles
Access time for array[2*4096]: 217 CPU cycles
Access time for array[3*4096]: 76 CPU cycles
Access time for array[4*4096]: 224 CPU cycles
Access time for array[5*4096]: 224 CPU cycles
Access time for array[6*4096]: 224 CPU cycles
Access time for array[7*4096]: 84 CPU cycles
Access time for array[8*4096]: 220 CPU cycles
Access time for array[9*4096]: 216 CPU cycles
accessing 2th time
Access time for array[0*4096]: 2082 CPU cycles
Access time for array[1*4096]: 225 CPU cycles
Access time for array[2*4096]: 217 CPU cycles
Access time for array[3*4096]: 91 CPU cycles
Access time for array[4*4096]: 216 CPU cycles
Access time for array[5*4096]: 222 CPU cycles
Access time for array[6*4096]: 213 CPU cycles
Access time for array[7*4096]: 89 CPU cycles
Access time for array[8*4096]: 222 CPU cycles
Access time for array[9*4096]: 219 CPU cycles
accessing 3th time
Access time for array[0*4096]: 2199 CPU cycles
Access time for array[1*4096]: 224 CPU cycles
Access time for array[2*4096]: 214 CPU cycles
Access time for array[3*4096]: 99 CPU cycles
Access time for array[4*4096]: 216 CPU cycles
Access time for array[5*4096]: 212 CPU cycles
Access time for array[6*4096]: 222 CPU cycles
Access time for array[7*4096]: 78 CPU cycles
Access time for array[7*4096]: 78 CPU cycles
Access time for array[8*4096]: 205 CPU cycles
Access time for array[9*4096]: 218 CPU cycles
accessing 4th time
Access time for array[0*4096]: 2075 CPU cycles
Access time for array[1*4096]: 232 CPU cycles
Access time for array[2*4096]: 234 CPU cycles
Access time for array[3*4096]: 117 CPU cycles
Access time for array[4*4096]: 245 CPU cycles
Access time for array[5*4096]: 234 CPU cycles
Access time for array[6*4096]: 240 CPU cycles
Access time for array[7*4096]: 123 CPU cycles
Access time for array[8*4096]: 236 CPU cycles
Access time for array[9*4096]: 324 CPU cycles
accessing 5th time
Access time for array[0*4096]: 2388 CPU cycles
Access time for array[1*4096]: 408 CPU cycles
Access time for array[2*4096]: 209 CPU cycles
Access time for array[3*4096]: 89 CPU cycles
Access time for array[4*4096]: 252 CPU cycles
Access time for array[5*4096]: 273 CPU cycles
Access time for array[6*4096]: 248 CPU cycles
Access time for array[7*4096]: 562 CPU cycles
Access time for array[8*4096]: 224 CPU cycles
Access time for array[9*4096]: 232 CPU cycles
accessing 6th time
Access time for array[0*4096]: 2239 CPU cycles
Access time for array[1*4096]: 214 CPU cycles
Access time for array[2*4096]: 219 CPU cycles
Access time for array[3*4096]: 87 CPU cycles
Access time for array[4*4096]: 197 CPU cycles
Access time for array[5*4096]: 345 CPU cycles
```

```
Access time for array[0*4096]: 2239 CPU cycles
Access time for array[1*4096]: 214 CPU cycles
Access time for array[2*4096]: 219 CPU cycles
Access time for array[3*4096]: 87 CPU cycles
Access time for array[4*4096]: 197 CPU cycles
Access time for array[5*4096]: 345 CPU cycles
Access time for array[6*4096]: 197 CPU cycles
Access time for array[7*4096]: 94 CPU cycles
Access time for array[8*4096]: 240 CPU cycles
Access time for array[9*4096]: 232 CPU cycles
accessing 7th time
Access time for array[0*4096]: 1697 CPU cycles
Access time for array[1*4096]: 208 CPU cycles
Access time for array[2*4096]: 158 CPU cycles
Access time for array[3*4096]: 74 CPU cycles
Access time for array[4*4096]: 156 CPU cycles
Access time for array[5*4096]: 175 CPU cycles
Access time for array[6*4096]: 193 CPU cycles
Access time for array[7*4096]: 72 CPU cycles
Access time for array[8*4096]: 148 CPU cycles
Access time for array[9*4096]: 478 CPU cycles
accessing 8th time
Access time for array[0*4096]: 2634 CPU cycles
Access time for array[1*4096]: 244 CPU cycles
Access time for array[2*4096]: 185 CPU cycles
Access time for array[3*4096]: 45 CPU cycles
Access time for array[4*4096]: 164 CPU cycles
Access time for array[5*4096]: 296 CPU cycles
Access time for array[6*4096]: 489 CPU cycles
Access time for array[7*4096]: 45 CPU cycles
Access time for array[8*4096]: 156 CPU cycles
Access time for array[9*4096]: 156 CPU cycles
Access time for array[3*4096]: 45 CPU cycles
Access time for array[4*4096]: 164 CPU cycles
Access time for array[5*4096]: 296 CPU cycles
Access time for array[6*4096]: 489 CPU cycles
Access time for array[7*4096]: 45 CPU cycles
Access time for array[8*4096]: 156 CPU cycles
Access time for array[9*4096]: 156 CPU cycles
accessing 9th time
Access time for array[0*4096]: 1349 CPU cycles
Access time for array[1*4096]: 178 CPU cycles
Access time for array[2*4096]: 170 CPU cycles
Access time for array[3*4096]: 58 CPU cycles
Access time for array[4*4096]: 312 CPU cycles
Access time for array[5*4096]: 261 CPU cycles
Access time for array[6*4096]: 295 CPU cycles
Access time for array[7*4096]: 183 CPU cycles
Access time for array[8*4096]: 318 CPU cycles
Access time for array[9*4096]: 294 CPU cycles
accessing 10th time
Access time for array[0*4096]: 1045 CPU cycles
Access time for array[1*4096]: 158 CPU cycles
Access time for array[2*4096]: 158 CPU cycles
Access time for array[3*4096]: 35 CPU cycles
Access time for array[4*4096]: 160 CPU cycles
Access time for array[5*4096]: 158 CPU cycles
Access time for array[6*4096]: 158 CPU cycles
Access time for array[7*4096]: 36 CPU cycles
Access time for array[8*4096]: 156 CPU cycles
Access time for array[9*4096]: 273 CPU cycles
```

(b) **Task 2: Using Cache as a Side Channel**: I ran the program `FlushReload.c` 20 times. Here are the results.

```
[03/16/23]seed@VM:~/.../Labsetup$ for i in {1..20}; do ./flushReload ; done | grep "Secret" | wc -l
441
[03/16/23]seed@VM:~/.../Labsetup$ for i in {1..20}; do ./flushReload ; done | grep "Secret" | wc -l
308
[03/16/23]seed@VM:~/.../Labsetup$ for i in {1..20}; do ./flushReload ; done | grep "Secret" | wc -l
196
[03/16/23]seed@VM:~/.../Labsetup$ for i in {1..20}; do ./flushReload ; done | grep "Secret" | wc -l
304
[03/16/23]seed@VM:~/.../Labsetup$ for i in {1..20}; do ./flushReload ; done | grep "Secret" | wc -l
626
```

It seems that the threshold **130** CPU cycles was too high, and a lot of other cache blocks fell under the threshold. Hence, I adjusted the threshold to **100** CPU cycles. Below is the result. I ran 25 times. I got the secret value 24 times, which was accurate, i.e., 94. Check `3.2_result` for complete log.

```
[03/16/23]seed@VM:~/.../Labsetup$ for i in {1..25}; do ./flushReload ; done > result
[03/16/23]seed@VM:~/.../Labsetup$ cat result | grep "Secret"
The Secret = 94.
The Secret = 94.
The Secret = 94.
The Secret = 94.
The Secret = 94.
The Secret = 94.
The Secret = 94.
The Secret = 94.
The Secret = 94.
The Secret = 94.
The Secret = 94.
The Secret = 94.
The Secret = 94.
The Secret = 94.
The Secret = 94.
The Secret = 94.
The Secret = 94.
The Secret = 94.
The Secret = 94.
The Secret = 94.
The Secret = 94.
The Secret = 94.
The Secret = 94.
The Secret = 94.
[03/16/23]seed@VM:~/.../Labsetup$ cat result | grep "Secret = 94" | wc -l
24
```

2. **Out-of-Order Execution and Branch Prediction**

   (a) I ran the spectre experiment 25 times, and I was able to read the secret value, i.e., **97**, 22 times. Hence, the line `temp = array[x*4096+DELTA]` is executed 22 times. To see the output, check `4.3_1_result`.

```
[03/16/23]seed@VM:~/.../Labsetup$ gcc -o spectreExp SpectreExperiment.c
[03/16/23]seed@VM:~/.../Labsetup$ for i in {1..25}; do ./spectreExp ; done > result
[03/16/23]seed@VM:~/.../Labsetup$ cat result | grep "Secret = 97" | wc -l
22
[03/16/23]seed@VM:~/.../Labsetup$ cat result | grep "Secret = 97"
The Secret = 97.
The Secret = 97.
The Secret = 97.
The Secret = 97.
The Secret = 97.
The Secret = 97.
The Secret = 97.
The Secret = 97.
The Secret = 97.
The Secret = 97.
The Secret = 97.
The Secret = 97.
The Secret = 97.
The Secret = 97.
The Secret = 97.
The Secret = 97.
The Secret = 97.
The Secret = 97.
The Secret = 97.
The Secret = 97.
The Secret = 97.
The Secret = 97.
The Secret = 97.
The Secret = 97.
[03/16/23]seed@VM:~/.../Labsetup$
```

   (b) After commenting the line that flushes size, below is the result. The secret value was seen only 13 times. To see the output, check `4.3_2_result`. The very likely reason for lesser

success is that `size` variable is now available in the cache (since the program did not flush it). It takes much lesser time to fetch it, so many times it was fetched before saving the secret value (`array[97*4096+DELTA]`) to the cache, and correct decision to not take the branch was made.

```
[03/16/23]seed@VM:~/.../Labsetup$ for i in {1..25}; do ./spectreExp ; done > result
[03/16/23]seed@VM:~/.../Labsetup$ cat result | grep "Secret = 97" | wc -l
13
[03/16/23]seed@VM:~/.../Labsetup$ cat result | grep "Secret = 97"
The Secret = 97.
The Secret = 97.
The Secret = 97.
The Secret = 97.
The Secret = 97.
The Secret = 97.
The Secret = 97.
The Secret = 97.
The Secret = 97.
The Secret = 97.
The Secret = 97.
The Secret = 97.
```

(c) Finally, by replacing the line `victim(i)` with `victim(i+20)`, we get no instances when we get the secret value **97**. Reason is, we trained the CPU to take the false branch, hence it never executes the true branch of the if-condition.

```
[03/16/23]seed@VM:~/.../Labsetup$ for i in {1..25}; do ./spectreExp ; done > result
[03/16/23]seed@VM:~/.../Labsetup$ cat result
[03/16/23]seed@VM:~/.../Labsetup$ cat result | grep "Secret = 97" | wc -l
0
```

3. **The Spectre Attack** I ran the task 10 times. All the times, the secret value was either 0 or 83 (char "S"). Since 0 is the value the program would normally give when correct branch is run, it is likely that the secret value is 83. See below for result. Also check `5_result` for the complete log.

```
[03/20/23]seed@VM:~/.../Labsetup$ for i in {1..10}; do ./spectreAttack; done > result
[03/20/23]seed@VM:~/.../Labsetup$ grep -ia "the secret" result
The Secret = 0().
The Secret = 83(S).
The Secret = 83(S).
The Secret = 0().
The Secret = 83(S).
The Secret = 0().
The Secret = 83(S).
The Secret = 0().
The Secret = 83(S).
The Secret = 0().
The Secret = 83(S).
The Secret = 0().
The Secret = 0().
The Secret = 83(S).
The Secret = 0().
The Secret = 83(S).
```

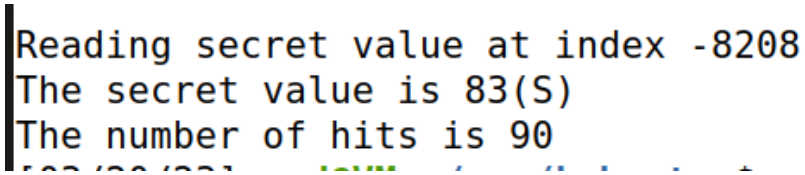
4. **Improve the Attack Accuracy**

(a) Indeed, it prints 0 as the secret value, as shown below.

```
Reading secret value at index -8208
The secret value is 0()
The number of hits is 919
[03/20/23]seed@VM:~/.../Labsetup$
```

The reason for printing 0 is that the CPU is able to correct the wrong decision of taking the true branch when false branch should be taken, most of the times. Hence, the frequency

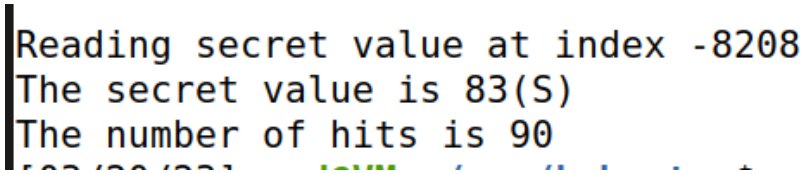
of hits for value 0 would be the most, hence we get 0 almost every time. We can avoid this by choosing the second max value according to hits, hence below is the result.



(b) Without the printf line, the attack does not succeed. The number of hits as 1 each time shows that the secret value seems to be completely random each time, so the highest score value is still 1.

```
[03/20/23]seed@VM:~/.../Labsetup$ for i in {1..10}; do ./spectreAttackImp; done
Reading secret value at index -8208
The secret value is 21()
The number of hits is 1
Reading secret value at index -8208
The secret value is 1()
The number of hits is 0
Reading secret value at index -8208
The secret value is 95(_)
The number of hits is 1
Reading secret value at index -8208
The secret value is 1()
The number of hits is 0
Reading secret value at index -8208
The secret value is 1()
The number of hits is 0
Reading secret value at index -8208
The secret value is 1()
The number of hits is 0
Reading secret value at index -8208
The secret value is 75(K)
The number of hits is 1
Reading secret value at index -8208
The secret value is 132(?)
The number of hits is 1
Reading secret value at index -8208
The secret value is 1()
The number of hits is 0
Reading secret value at index -8208
The secret value is 165(?)
The number of hits is 1
```

With the printf line, the attack works the first time, as shown before.



Upon further investigation, it seems that the attack is only successful when at least two characters or new lines are printed. Print 1 character (or new line), the attack would not work. Further, if the printf statement is moved out of the loop that runs 1000 times, the above observation still applies. My initial guess is, the address of the string in printf is stored somewhere on the memory and its position on the memory is essential to make the attack happen, let's say to make the `index_beyond` as exactly the value required to reach the secret data. It might also be related to padding issue, but I am not sure.

(c) To check the success rate depending on how long program sleeps, I ran the program with following numbers in microseconds: **0, 10, 50, 100, 1000**.

sleep 0: Seems that hits for secret value 83 are very low, and for once, secret value is 2.

```
[03/20/23]seed@VM:~/.../Labsetup$ grep -ia "secret\|number" result
Reading secret value at index -8208
The secret value is 83(S)
The number of hits is 9
Reading secret value at index -8208
The secret value is 83(S)
The number of hits is 3
Reading secret value at index -8208
The secret value is 2()
The number of hits is 83
Reading secret value at index -8208
The secret value is 83(S)
The number of hits is 7
Reading secret value at index -8208
The secret value is 83(S)
The number of hits is 3
Reading secret value at index -8208
The secret value is 83(S)
The number of hits is 5
Reading secret value at index -8208
The secret value is 83(S)
The number of hits is 2
Reading secret value at index -8208
The secret value is 83(S)
The number of hits is 6
Reading secret value at index -8208
The secret value is 83(S)
The number of hits is 8
Reading secret value at index -8208
The secret value is 83(S)
The number of hits is 2
[03/20/23]seed@VM:~/.../Labsetup$
```

sleep 10: Improvement on the hits and all secret values are 83.

```
[03/20/23]seed@VM:~/.../Labsetup$ grep -ia "secret\|number" result
Reading secret value at index -8208
The secret value is 83(S)
The number of hits is 21
Reading secret value at index -8208
The secret value is 83(S)
The number of hits is 34
Reading secret value at index -8208
The secret value is 83(S)
The number of hits is 14
Reading secret value at index -8208
The secret value is 83(S)
The number of hits is 3
Reading secret value at index -8208
The secret value is 83(S)
The number of hits is 15
Reading secret value at index -8208
The secret value is 83(S)
The number of hits is 13
Reading secret value at index -8208
The secret value is 83(S)
The number of hits is 25
Reading secret value at index -8208
The secret value is 83(S)
The number of hits is 24
Reading secret value at index -8208
The secret value is 83(S)
The number of hits is 9
Reading secret value at index -8208
The secret value is 83(S)
The number of hits is 4
[03/20/23]seed@VM:~/.../Labsetup$
```

sleep 50: Little improvement from before on hits, and all secret values are 83.

```
[03/20/23]seed@VM:~/.../Labsetup$ grep -ia "secret\|number" result
Reading secret value at index -8208
The secret value is 83(S)
The number of hits is 41
Reading secret value at index -8208
The secret value is 83(S)
The number of hits is 13
Reading secret value at index -8208
The secret value is 83(S)
The number of hits is 18
Reading secret value at index -8208
The secret value is 83(S)
The number of hits is 27
Reading secret value at index -8208
The secret value is 83(S)
The number of hits is 29
Reading secret value at index -8208
The secret value is 83(S)
The number of hits is 14
Reading secret value at index -8208
The secret value is 83(S)
The number of hits is 35
Reading secret value at index -8208
The secret value is 83(S)
The number of hits is 28
Reading secret value at index -8208
The secret value is 83(S)
The number of hits is 18
Reading secret value at index -8208
The secret value is 83(S)
The number of hits is 24
[03/20/23]seed@VM:~/.../Labsetup$
```

sleep 100: Hits almost same as before, and all secret values are 83.

```
[03/20/23]seed@VM:~/.../Labsetup$ grep -ia "secret\|number" result
Reading secret value at index -8208
The secret value is 83(S)
The number of hits is 18
Reading secret value at index -8208
The secret value is 83(S)
The number of hits is 5
Reading secret value at index -8208
The secret value is 83(S)
The number of hits is 16
Reading secret value at index -8208
The secret value is 83(S)
The number of hits is 27
Reading secret value at index -8208
The secret value is 83(S)
The number of hits is 23
Reading secret value at index -8208
The secret value is 83(S)
The number of hits is 9
Reading secret value at index -8208
The secret value is 83(S)
The number of hits is 27
Reading secret value at index -8208
The secret value is 83(S)
The number of hits is 14
Reading secret value at index -8208
The secret value is 83(S)
The number of hits is 4
Reading secret value at index -8208
The secret value is 83(S)
The number of hits is 12
[03/20/23]seed@VM:~/.../Labsetup$
```

sleep 1000: Significant improvement on the hits, and all secret values are 83.

```
[03/20/23]seed@VM:~/.../Labsetup$ grep -ia "secret\|number" result
Reading secret value at index -8208
The secret value is 0()
The number of hits is 91
Reading secret value at index -8208
The secret value is 83(S)
The number of hits is 15
Reading secret value at index -8208
The secret value is 83(S)
The number of hits is 104
Reading secret value at index -8208
The secret value is 83(S)
The number of hits is 93
Reading secret value at index -8208
The secret value is 83(S)
The number of hits is 81
Reading secret value at index -8208
The secret value is 0()
The number of hits is 96
Reading secret value at index -8208
The secret value is 83(S)
The number of hits is 105
Reading secret value at index -8208
The secret value is 0()
The number of hits is 105
Reading secret value at index -8208
The secret value is 83(S)
The number of hits is 86
Reading secret value at index -8208
The secret value is 0()
The number of hits is 100
[03/20/23]seed@VM:~/.../Labsetup$
```

It seems that increasing the length of sleep improves the chances of getting the right value.

5. **Steal the Entire Secret String** I created a new program `SpectreAttackImprovedWhole.c` by adapting `SpectreAttackImproved.c` for the whole string. Also, for each character, I ran the attack, for each character, 10000 times (as opposed to 1000 times before) to hope to get a better accuracy. Below is the result, hence the string "Some Secret Value" is very likely our secret string. Check `7_result` for the whole log. It is interesting to note that this task even works when the mysterious printf (that was talked about earlier) is commented out.

```
[03/20/23]seed@VM:~/.../Labsetup$ gcc -o spectreAttackImpWhole SpectreAttackImprovedWhole.c
[03/20/23]seed@VM:~/.../Labsetup$ ./spectreAttackImpWhole > result
[03/20/23]seed@VM:~/.../Labsetup$ grep -ia "the secret" result
The secret value at position 0 is 83(S)
The secret value at position 1 is 111(o)
The secret value at position 2 is 109(m)
The secret value at position 3 is 101(e)
The secret value at position 4 is 32( )
The secret value at position 5 is 83(S)
The secret value at position 6 is 101(e)
The secret value at position 7 is 99(c)
The secret value at position 8 is 114(r)
The secret value at position 9 is 101(e)
The secret value at position 10 is 116(t)
The secret value at position 11 is 32( )
The secret value at position 12 is 86(V)
The secret value at position 13 is 97(a)
The secret value at position 14 is 108(l)
The secret value at position 15 is 117(u)
The secret value at position 16 is 101(e)
```