# Speculation

This document explains potential effects of speculation, and how undesirable effects can be mitigated portably using common APIs.

---

To improve performance and minimize average latencies, many contemporary CPUs employ speculative execution techniques such as branch prediction, performing work which may be discarded at a later stage.

Typically speculative execution cannot be observed from architectural state, such as the contents of registers. However, in some cases it is possible to observe its impact on microarchitectural state, such as the presence or absence of data in caches. Such state may form side-channels which can be observed to extract secret information.

For example, in the presence of branch prediction, it is possible for bounds checks to be ignored by code which is speculatively executed. Consider the following code:

```
int load_array(int *array, unsigned int index)
{
        if (index >= MAX_ARRAY_ELEMS)
                return 0;
        else
                return array[index];
}
```

Which, on arm64, may be compiled to an assembly sequence such as:

```
      CMP     <index>, #MAX_ARRAY_ELEMS
      B.LT    less
      MOV     <returnval>, #0
      RET
less:
      LDR     <returnval>, [<array>, <index>]
      RET
```

It is possible that a CPU mis-predicts the conditional branch, and speculatively loads array[index], even if index >= MAX_ARRAY_ELEMS. This value will subsequently be discarded, but the speculated load may affect microarchitectural state which can be subsequently measured.

More complex sequences involving multiple dependent memory accesses may result in sensitive information being leaked. Consider the following code, building on the prior example:

```
int load_dependent_arrays(int *arr1, int *arr2, int index)
{
        int val1, val2,

        val1 = load_array(arr1, index);
        val2 = load_array(arr2, val1);

        return val2;
}
```

Under speculation, the first call to load_array() may return the value of an out-of-bounds address, while the second call will influence microarchitectural state dependent on this value. This may provide an arbitrary read primitive.

## Mitigating speculation side-channels

The kernel provides a generic API to ensure that bounds checks are respected even under speculation. Architectures which are affected by speculation-based side-channels are expected to implement these primitives.

The array_index_nospec() helper in <linux/nospec.h> can be used to prevent information from being leaked via side-channels.

A call to array_index_nospec(index, size) returns a sanitized index value that is bounded to [0, size) even under cpu speculation conditions.

This can be used to protect the earlier load_array() example:

```
int load_array(int *array, unsigned int index)
{
        if (index >= MAX_ARRAY_ELEMS)
                return 0;
        else {
                index = array_index_nospec(index, MAX_ARRAY_ELEMS);
                return array[index];
        }
}
```