# Inconsistent null check of pointer

**Name:** Inconsistent null check of pointer

**Description:** A dereferenced pointer is not checked for nullness in this location, but it is checked in other locations. Dereferencing a null pointer leads to undefined results.

**ID:** cpp/inconsistent-nullness-testing

**Kind:** problem

**Severity:** warning

This query finds pointer dereferences that do not first check the pointer for nullness, even though the same pointer is checked for nullness in other parts of the code. It is likely that the nullness check was accidentally omitted, and that a null pointer dereference can occur. Dereferencing a null pointer and attempting to modify its contents can lead to anything from a segmentation fault to corrupting important system data (including the interrupt table in some architectures).

This check is an approximation, so some results may not be actual defects in the program. It is not possible in general to compute the values of pointers without running the program with all input data.

## Recommendation

Use a nullness check consistently in all cases where a pointer is dereferenced.

## Example

This code shows two examples where a pointer is dereferenced. The first example checks that the pointer is not null before dereferencing it. The second example fails to perform a nullness check, leading to a potential vulnerability in the code.

```
1  void* f() {
2      block = (MyBlock *)malloc(sizeof(MyBlock));
3      if (block) { //correct: block is checked for nullness here
4          block->id = NORMAL_BLOCK_ID;
5      }
6      //...
7      /* make sure data-portion is null-terminated */
8      block->data[BLOCK_SIZE - 1] = '\0'; //wrong: block not checked for nullness here
9      return block;
10 }
```

## References

- SEI CERT C++ Coding Standard: MEM10-C. Define and use a pointer validation function.
- Common Weakness Enumeration: CWE-476.