rules.sonarsource.com

C++ static code analysis: Track parsing failures

3 minutes

When the analysis succeeds, it doesn't mean that the analyzer was able to understand all the analyzed code. If the analyzer fails to parse some parts of your code, it will ignore them during the analysis. This rule will help you track these parsing failures.

Why they happen:

There are many reasons why parsing failures can happen, here are the common ones:

- Compiler extensions: Your compiler might allow you to write code that isn't standard-conforming.
- Bad analysis environment. This usually means that the environment during the build is different than the one during the analysis. For example, files or symbolic links that were available during the build are not available during the analysis.
- Use of new language features that are not yet supported by our analyzer.
- Limitation in our analyzer. We are always working on improving this.

How they impact analysis:

We cannot judge without looking at specific examples, as they contain a broad range of types of errors. On our side, we will make sure that you get notified through the analysis logs when they have an impact on the quality of the analysis.

How to deal with them:

There are three recommended ways to deal with parsing failures:

- Fix them when it is possible. It should be obvious from the message if you can do it. For example, by replacing the use of a compiler extension with the standard-conforming equivalent.
- If you cannot fix them and the analysis logs state that they have a bad impact on the analysis results, Report them.
- If you cannot fix them and the analysis logs don't state anything explicit about their impact, ignore them by resolving them as "won't fix".

Noncompliant Code Example

```
// This example uses Microsoft extension /Zc:referenceBinding that
allows binding r-value to l-value. Even though your compiler might
allow it, our analyzer will flag it

struct S {
...
};

void f(S&) {
...
}

int main() {
  f(S{}); // Noncompliant: no matching function for call to 'f'
}
```

Compliant Solution

```
// Here we are showing how to fix the issue by replacing the code
relying on a compiler extension by standard-conforming equivalent

struct S {
...
};

void f(S&&) { // Using C++11 r-value reference fixes the issue
...
```

```
}

int main() {
  f(S{}); // Compliant
}
```