

+ 25

Inherently Unsafe

Practical Approaches for Writing Safer C++

ASSAF TZUR-EL



20
25



Reminder:
Slides != Talk
To get the whole picture,
watch the session.

“ C makes it easy
to shoot yourself in the foot;
C++ makes it harder,
but when you do –
it blows your whole leg off.

Bjarne Stroustrup

“ C makes it easy
to shoot yourself in the foot;
C++
but
it blows you off.

```
const int ANSWER = 42;  
int *p = &ANSWER;  
*p = 43;
```

warning C4090: 'initializing': different 'const' qualifiers

Bjarne Stroustrup

“ C makes it easy
to shoot yourself in the foot;
C++
but
it blows you off.

```
const int ANSWER = 42;  
int *p = &ANSWER;  
*p = 43;
```

error C2440: 'initializing': cannot convert from 'const int *' to 'int *'

Bjarne Stroustrup

“ C makes it easy
to shoot yourself in the foot;

```
const int ANSWER = 42;  
const_cast<int &>(ANSWER) = 43;
```

it blows your whole leg off.

Bjarne Stroustrup

Keeping our legs

The dangers of C++

- Safety
 - 🤪
- Unpredictable behavior
 - 🤪

Safety

```
void process_data(const char *input)
{
    char buffer[SIZE_LIMIT];

    strcpy(buffer, input);
    modify(buffer);
}
```

Safety Risks

- Lifetime safety
- Bounds safety
- Type safety
- Thread safety
- Runtime checks
- Combinations

Unpredictability

- Undefined behavior

```
int result = num / denum;
```

Unpredictability

- Undefined behavior
- Unspecified behavior

```
void f(int x, int y, int z);  
f(a(), b(), c());
```

Unpredictability

- Undefined behavior
- Unspecified behavior
- Implementation-defined behavior

```
sizeof(long);
```

Unpredictability

- Undefined behavior
- Unspecified behavior
- Implementation-defined behavior
- More

invalidates pointers, iterators, and references

conditionally-supported

program construct that an implementation is not required to support

locale-specific behavior

behavior that depends on local conventions of nationality, culture, and language

The answer

- Switch to a safer language!
- CISA, NSA, ONCD's goal
- Not practical
 - Existing code base
 - Existing ecosystem
 - Requirements
 - No alternative

~~The answer~~

- ~~Switch to a safer language!~~
- ~~CISA, NSA, ONCD's goal~~
- ~~Not practical~~
 - ~~Existing code base~~
 - ~~Existing ecosystem~~
 - ~~Requirements~~
 - ~~No alternative~~

Solutions

- Standardization
 - C++ standard
 - Safe C++
 - ISO 26262
- Coding practices
 - OWASP Top Ten
 - Defensive programming
 - Secure by Design

Enforcement

- Rules
 - MISRA
- Tools

Motor Industry Software Reliability Association



MISRA C++:2023

Guidelines for the use of
C++17 in critical systems

October 2023



Example

Rule 9.6.1 The **goto** statement should not be used

Category Advisory

Analysis Decidable, Single Translation Unit

Rationale

The use of **goto** is usually regarded as bad programming practice as it can lead to code that is difficult to understand and analyse. Restructuring code to avoid its use generally leads to code that has a lower level of complexity.

Another example

Rule 9.4.2 The structure of a **switch** statement shall be appropriate

| | | |
|----------|------------------------------------|---|
| Category | Required | [stmt.switch] [dcl.attr.fallthrough] |
| Analysis | Decidable, Single Translation Unit | |

Another example

Rule 9.4.2 The structure of a **switch** statement shall be appropriate

Category Required [stmt.switch]
[dcl.attr.fallthrough]

Analysis Decidable, Single Translation Unit

Amplification

A **switch** statement is structured appropriately when it conforms to the following restrictions:

1. The *condition* shall only be preceded by an optional *simple-declaration*;
...
7. Every **switch** statement shall have a **default** label, appearing as either the first label of the first *switch label group* or as the last label of the last *switch label group*.

Yes, but...

Rule 0.0.1 A function shall not contain *unreachable* statements

[IEC 61508-7] / C.5.9

[DO-178C] / 6.4.4.3.c

[ISO 26262-6] / 9.4

Category Required

Analysis Decidable, Single Translation Unit

Tools

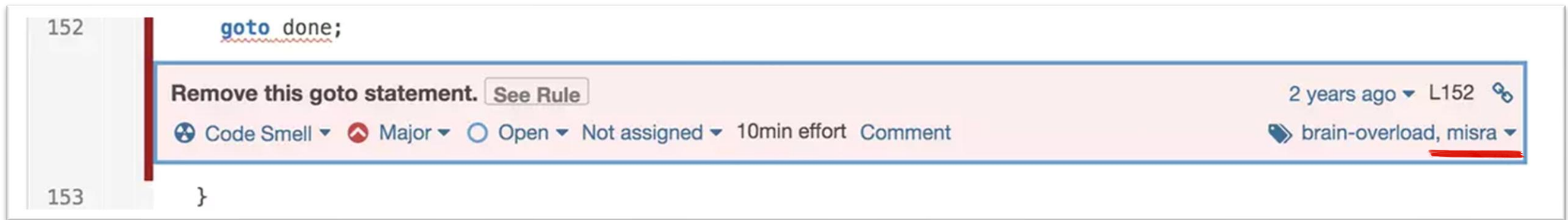


Image courtesy of ALM Toolbox, a Sonar partner

Tools

```
181     else
182     {
183     #ifdef DEBUG_OFF
184         /*
185          * NULL is only allowed in debug mode, since the sidno does not
186          * make sense for users but is useful to include in debug
187          * printouts. Therefore, we want to ASSERT(0) in non-debug mode.
188          * Since there is no ASSERT in non-debug mode, we use abort
189          * instead.
190          */
191         abort();
192     #endif
193     ret= sprintf(buf, "%d:%lld", sidno, gno);
```

code will never be executed [See Rule](#)

2 years ago ▾ L193 

 Bug ▾

 Major ▾

 Open ▾

Not assigned ▾

5min effort [Comment](#)

 cert, cwe, misra, unused ▾

Image courtesy of ALM Toolbox, a Sonar partner

The Alternative: C++ Core Guidelines

- Stroustrup's approach
- Core Guidelines: Modern C++ best practices
- Profiles: tailored language subsets
- Tool-enforced: static analysis and compilers
- Balance: safety without sacrificing performance

MISRA vs. Core Guidelines

| MISRA C++ | C++ Core Guidelines |
|------------------------------|---------------------------|
| Strict subset | Flexible profiles |
| Safety-critical focus | General-purpose |
| Forbids unsafe features | Discourages unsafe use |
| Compliance and audits | Tool-enforced |
| Conservative, slow to evolve | Evolves with the standard |

Why?

- Our goal: Working software, not writing code
- Code quality: Reliability and Maintainability
- Safety
- Portability
- Code reviews and audits
- Quality and standards

Why?

“ C makes it easy
to shoot yourself in the foot;
C++ makes it harder,
but when you do –
it blows your whole leg off.

Bjarne Stroustrup

Action Items

- Where could your C++ code surprise you?
- What are you going to do about it?

Inherently Unsafe | Assaf Tzur-El



<https://www.linkedin.com/in/assaftzurel/>



assaf@tzurel.co.il



<https://wa.me/972543330085>



Eran Gilad, Gilad Darmon, Nir Dobovizki

Questions?