Establishing Coding Requirements for Non-Safety-Critical C++ Systems

Writing secure C++ code is hard and existing coding standards are insufficient. Our research focuses on educating developers about C++ security issues through quality secure coding rules and alerting developers of security-related deficiencies in their source code through automated checkers.

The CERT C++ Coding Standard comprises 83 C++-specific rules spread over 11 broad categories of language constructs. Additionally, the Standard references 79 (out of the 102) rules from the CERT C Coding Standard that also apply to C++. Each rule has a title, introduction & normative text, followed by a series of noncompliant code examples and their accompanying compliant solutions. Each rule also guides the user to the risks of failing to comply with the rule, what kind of automated detection mechanisms exist, what real-world vulnerabilities have resulted from failing to comply with the rule, and citations & related material.

Modified **137** C++-related rules and created an additional **16** rules on our public Wiki, engaging an average of **2000** unique visits per month. Contributed **15** checkers to the Clang open source C/C++ compiler, **available by default for 10s of millions of programmers.**

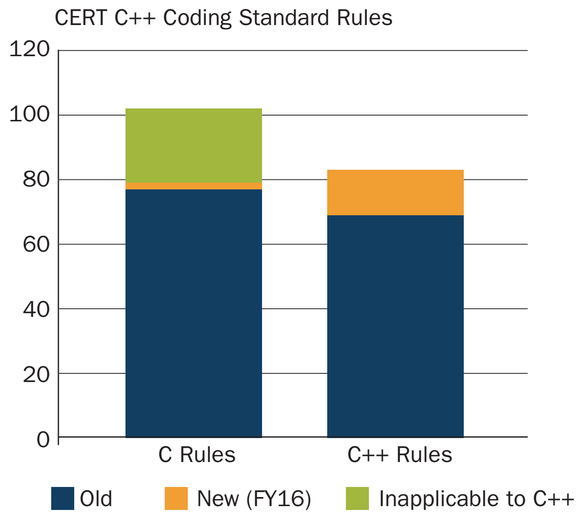
Research the kernel of a security-focused rule

Rule creation follows an iterative process involving multiple parties:

Hackers, authors, the C++ committee, and the C++ Standard itself help form the kernel of a rule. External collaborators such as compiler writers and users help iterate the rule concept and checker behavior until it is solid and applicable to real-world code.

The results are a more compelling rule and automatic detection capabilities.

Our Results: Rules



Our Results: Sections

1. Declarations and Initialization (DCL)
2. Expressions (EXP)
3. Integers (INT)
4. Containers (CTR)
5. Characters and Strings (STR)
6. Memory Management (MEM)
7. Input Output (FIO)
8. Exceptions and Error Handling (ERR)
9. Object Oriented Programming (OOP)
10. Concurrency (CON)
11. Miscellaneous (MSC)

JTC1/SC22/WG21 – The C++ Standards Committee

- Effective C++ Third Edition
- ISO
- IEC
- Common Vulnerabilities and Exposures
- Clang is the primary compiler for XCode and is thus used to build all iOS and MacOS applications, as well as FreeBSD. And is supported by Microsoft Visual Studio and Linux.

Create the stub rule text

ERR52-CPP Do not use setjmp() or longjmp()

Created by Fred Long, last modified by Sandy Shrum about 2 hours ago

The C standard library facilities `setjmp()` and `longjmp()` can be used to simulate throwing and catching exceptions. However, these facilities bypass automatic resource management and can result in undefined behavior, commonly including resource leaks, and denial-of-service attacks.

Create a basic checker for the rule text

```
E:llvm\2015>clang-tidy -checks=-*,cert-*
E:|Desktop\test1.cpp -- - std=c++14
2 warnings generated.
E:\Desktop\test1.cpp:7:7: warning: do not call 'setjmp';
consider using exception handling instead
[cert-err52-cpp] if (setjmp(env) == 0) {
```

Finish with a compelling rule that is applicable to realworld code and can be automatically enforced

Example Rule

DCL22-CPP. Functions declared with [[noreturn]] must return void

Created by Aaron Ballman, last modified on Aug 24, 2016

As described in MSC55-CPP. Do not return from a function declared [[noreturn]], functions declared with the `[[noreturn]]` attribute must not return on any code path. If a function declared with the `[[noreturn]]` attribute has a non-`void` return value, it implies that the function returns a value to the caller even though it would result in undefined behavior. Therefore, functions declared with `[[noreturn]]` must also be declared as returning `void`.

Noncompliant Code Example

In this noncompliant code example, the function declared with `[[noreturn]]` claims to return an `int`:

```
#include <cstdlib>

[[noreturn]] int f() {
  std::exit(0);
  return 0;
}
```

This example does not violate MSC55-CPP. Do not return from a function declared [[noreturn]] because `std::exit()` is declared `[[noreturn]]`, so the `return 0;` statement can never be executed.

Compliant Solution

Because the function is declared `[[noreturn]]`, and no code paths in the function allow for a return in order to comply with MSC55-CPP. Do not return from a function declared [[noreturn]], the compliant solution declares the function as returning `void` and elides the explicit `return` statement:

```
#include <cstdlib>

[[noreturn]] void f() {
  std::exit(0);
}
```

Risk Assessment

A function declared with a non-`void` return type and declared with the `[[noreturn]]` attribute is confusing to consumers of the function because the two declarations are conflicting. In turn, it can result in misuse of the API by the consumer or can indicate an implementation bug by the producer.

Rule	Severity	Likelihood	Remediation Cost	Priority	Level
DCL22-CPP	Low	Unlikely	Low	P3	L3

Automated Detection

Tool	Version	Checker	Description
Clang	3.9	`-Winvalid-noreturn`	

Related Vulnerabilities

Search for vulnerabilities resulting from the violation of this rule on the CERT website.

Related Guidelines

SEI CERT C++ Coding Standard	MSC54-CPP. Value-returning functions must return a value from all exit paths MSC55-CPP. Do not return from a function declared [[noreturn]]

Bibliography

[ISO/IEC 14882-2014]	Subclause 7.6.3, "Noreturn Attribute"

Contact: Aaron Ballman | aaron@aaronballman.com