Proposal for C2x
WG14 N2215

Title:	The `nodiscard` attribute
Author, affiliation:	Aaron Ballman, GrammaTech
Date:	2018-03-19
Proposal category:	New features
Target audience:	General developers, compiler/tooling developers

Abstract: Many functions return a value, however, not all function return values are of equal importance to the caller. There is insufficient information encoded within a function declaration to determine whether ignoring the return value at a call site is erroneous or not, and implementations generally do not diagnose that situation by default due to a very high false-positive rate. However, the designer of that function generally has sufficient knowledge of the problem domain to determine whether ignoring the results of the function call is a likely sign of a programming mistake. This paper proposes a new attribute, `nodiscard`, to convey this information.

Prior art: C++ has this feature using the same syntax. Various vendors have vendor-specific extensions such as `__attribute__((warn_unused_result))` that provide a similar mechanism.

The nodiscard attribute

Reply-to: Aaron Ballman (aaron@aaronballman.com; aballman@grammatech.com)
Document No: N2215
Date: 2018-03-19

Summary of Changes

N2215

- Added the appropriate cover page.
- Rebased on top of N2165, the latest attributes syntax proposal

N2051

- Original proposal.

Introduction

Many functions return a value, however, not all function return values are of equal importance to the caller. Instead, function return values tend to fall into one of three high-level categories: crucial (such as the value returned by a call to `malloc()`), informative-and-sometimes-important (such as the value returned by a call to `sprintf()`), or purely informative (such as the value returned by a call to `printf()`). A crucial return value is one which likely represents a programming error if it is ignored by the caller. There is insufficient information encoded within a function declaration to determine whether ignoring the return value at a call site is erroneous or not, and implementations generally do not diagnose that situation by default due to a very high false-positive rate. However, the designer of that function generally has sufficient knowledge of the problem domain to determine whether ignoring the results of the function call is a likely sign of a programming mistake.

Rationale

The `[[nodiscard]]` attribute has extensive real-world use, being implemented by Clang and GCC as `__attribute__((warn_unused_result))`, but was standardized under the name `[[nodiscard]]` by WG21. This proposal chose the identifier `nodiscard` because deviation from this name would create a needless incompatibility with C++.

The semantics of this attribute rely heavily on the notion of a use, the definition of which is left to implementation discretion. However, the non-normative guidance specified by WG21 is to encourage implementations to emit a warning diagnostic when a nodiscard function call is used in a potentially-evalulated discarded-value expression unless it is an explicit cast to void. This means that an implementation is not encouraged to perform dataflow analysis (like an initialized-but-unused local variable diagnostic would require). e.g.,

```
#define IGNORE(X)  ((void)X)

[[nodiscard]] int foo(void);
void func(void) {
  foo(); // Diagnose

  (void)foo(); // Do not diagnose
  IGNORE(foo()); // Do not diagnose

  _Generic(foo(), default : 1); // Do not diagnose (foo()
                                // is not evaluated).

  int i = foo();
  // Do not diagnose, even though i is never used.
}
```

This proposal does not cover applying the `[[nodiscard]]` attribute to a typedef type because the WG21 attribute does not apply to such a type. However, this may be a reasonable, motivating use case for the attribute given the common design patterns using typedef in C. If WG14 determines that this attribute should appertain to typedefs as well, it would be reasonable to propose this addition to WG21 as well to avoid incompatibilities between the two languages.

Proposal

This document proposes the `[[nodiscard]]` attribute as a way for an API designer to specify intent, allowing implementations to diagnose a function call expression that implicitly discards the result value when that result is crucial to correctly using the API.

The `[[nodiscard]]` attribute can be applied to the identifier in a function declarator to encourage implementations to diagnose ignoring the return value of a call to the function. As a motivating example, consider:

```
const size_t kNeededSize = ...; // Always > 0

void func(void *ptr, size_t size) {
  // If size is insufficient for our needs, resize the
  // given buffer to the larger size.
  if (size < kNeededSize) {
    realloc(ptr, kNeededSize); // (1)
    // (2)
  }

  int *i_ptr = (int *)ptr;
  for (size_t i = 0; i < (kSizeNeeded / sizeof(int)); ++i) {
    i_ptr[i] += 12; // (3)
  }
}
```

The call to `realloc()` at (1) attempts to resize the given buffer pointed to by ptr. However, this code has two potential security-related failures:

If `realloc()` fails to resize the buffer, that information is lost at (2) and `ptr` retains its original size. This means that `ptr` points to a buffer of insufficient size and when (3) is executed, a buffer overrun occurs. If `realloc()` fails to resize the buffer in place, then the pointer pointed to by `ptr` has been freed and when (3) is executed, it writes to freed memory. Either situation can lead to an exploitable security vulnerability [CWE-131, CWE-416]. (Note that CWE-131 is a CWE Top 25 vulnerability.)

If `realloc()` were instead declared `[[nodiscard]] void *realloc(void *, size_t);`, then implicitly discarding the return value at (1) can be diagnosed with an on-by-default warning, without inundating the user with false-positives elsewhere.

Additionally, the `[[nodiscard]]` attribute can be applied to the declaration of a struct, union, or enumeration type that, when used as the return type of a function, implies the function results should not be discarded. For instance, a code author may decide to annotate the declaration of a custom `critical_error_information` struct type with `[[nodiscard]]` rather than mark each individual function declaration returning `critical_error_information`. This eliminates the chance that a function returning such a type accidentally allows the results to be discarded by failing to write the attribute on a function declaration.

Proposed Wording

This proposed wording currently uses placeholder terms of art and it references a new subclause from WG14 N2165, 6.7.11, Attributes that describes the referenced grammar terms. The [Note] in paragraph 1 of the semantics is intended to convey informative guidance rather than normative requirements.

Add new subclause:

6.7.11.4 Nodiscard attribute

Constraints

1 The *attribute-token* `nodiscard` shall be applied to the *identifier* in a function declarator or to the definition of a structure, union, or enumeration type. It shall appear at most once in each *attribute-list* and no *attribute-argument-clause* shall be present.

Semantics

1 [Note: A nodiscard call is a function call expression that calls a function previously declared `nodiscard`, or whose return type is a structure, union, or enumeration type marked `nodiscard`. Evaluation of a nodiscard call as a void expression (6.8.3) is discouraged unless explicitly cast to `void`. Implementations are encouraged to issue a warning in such cases. This is typically because discarding the return value of a nodiscard call has surprising consequences. -- end note]

2 EXAMPLE

```
struct [[nodiscard]] error_info { /*...*/ };
struct error_info enable_missile_safety_mode(void);
void launch_missiles(void);
void test_missiles(void) {
  enable_missile_safety_mode(); // warning encouraged
  launch_missiles();
}
```

Acknowledgements

I would like to recognize the following people for their help in this work: David Keaton, David Svoboda, and Andrew Tomazos.

References

[N2165]
Attributes in C. Aaron Ballman. http://www.open-std.org/jtc1/sc22/wg14/www/docs/n2165.pdf

[P0068R0]
Proposal of [[unused]], [[nodiscard]] and [[fallthrough]] attributes. Andrew Tomazos. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/p0068r0.pdf

[P0189R1]
Wording for [[nodiscard]] attribute. Andrew Tomazos. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2016/p0189r1.pdf

[CWE-131]
CWE-131: Incorrect Calculation of Buffer Size. http://cwe.mitre.org/data/definitions/131.html

[CWE-416]
CWE-416: Use After Free. http://cwe.mitre.org/data/definitions/416.html