C++ ALLIANCE FELLOWSHIP · MID-TERM · JULY 2026

Enhance Clang Diagnostics

Presenter

Aditya · flash1729

Mentors

Vassil Vassilev · Aaron Jomy

OVERVIEW

What I worked on

01 · #46362

Function references compared to null

merged

A comparison that's always true, and Clang wasn't saying so. Nine lines in Sema.

02 · WG14 N3410

Mixed internal & external linkage in C

reland in final review

Undefined behavior since C89, reported in 2022, and Clang compiled it quietly. A new diagnostic.

03 · #202945

`-Wunused-template` under `-Wall`

merged

Clean the whole monorepo, then turn the warning on so the pattern can't come back.

ISSUE 01 · CLANG SEMA · #46362

A comparison that's always true, and no warning

-Wtautological-pointer-compare

Direct comparison, Clang warns

```
void foo();
if (foo != nullptr) {}
// warning: always true
// [-Wtautological-pointer-compare]
```

Through a reference, nothing

```
void (&f)() = foo;
if (f != nullptr) {}
// no warning (the bug)
// same flag should fire
```

A function reference can't be null either, so this comparison is just as always-true and deserved the same `-Wtautological-pointer-compare` warning. The bool-conversion path (`-Wpointer-bool-conversion`) had the same blind spot.

ISSUE 01 · CLANG SEMA · PR #204944

Looking through the reference

01 `DiagnoseAlwaysNonNullPointer` saw `void (&)()`, a reference rather than a function, and bailed out early.

02 Look through the reference to the function type beneath, so the existing check treats it like a function.

03 Skip the `&`-prefix fix-it for references, since it wouldn't actually silence the warning there.

04 In review: added a release note, merged two adjacent `if`s, and rebased onto main.

status: merged

~9 lines

in SemaChecking.cpp

```
void (&)()
↓ look through the reference
void ()
↓ existing check now fires
warning: always true
```

Tests cover both warning paths plus a negative case (a reference to a pointer can be null). The full suite flagged two tests that now warn correctly; their expectations were updated.

ISSUE 02 · WG14 N3410 · #54215

One name ends up with two linkages

```
static int x;       // internal
void f(void) {
  int x;            // no linkage
  {
    extern int x;   // external!
  }
}
```

The local `x` shadows the static. The inner `extern` inherits from the visible declaration, which has no linkage, so it falls back to external (C11 6.2.2p4).

One identifier ends up with **internal and external linkage in the same translation unit**. That's undefined behavior from C89 through C23 (6.2.2p7).

GCC already rejected this. Clang compiled it quietly, since 2022.

N3410 (Uecker) makes it a hard constraint violation in C2y.

ISSUE 02 · WG14 N3410

Why C and C++ disagree

C: conflict

Lookup for the `extern` respects the shadow.

Linkage inheritance breaks, the extern falls back to external, and internal meets external in one translation unit.

C++: no conflict

A block-scope `extern` targets the enclosing namespace scope ([dcl.meaning.general]/3.5).

It walks past the shadow, finds the `static`, and inherits internal linkage.

So the diagnostic has to fire **only for C**. The C++ path is legal by design.

ISSUE 02 · WG14 N3410 · PR #193567

The signal was already sitting in Sema

`LookupResult::isShadowed()` already knew lookup had skipped a shadowing declaration. No new state needed; the signal was sitting in Sema all along.

- ✓ Language is C, not C++
- ✓ New declaration is a block-scope `extern`
- ✓ Lookup was shadowed
- ✓ Prior declaration has internal linkage

A reviewer caught a refinement: dropping an `isFileVarDecl()` check fixed a false negative on chained shadows, where the prior internal-linkage declaration is itself a block-scope extern. Hard error in C2y per N3410; a "behavior is undefined" note in earlier modes via one `%select`.

status: reland in final review, CI green

+169 / −31

across 6 files

N3410 conformance claimed for **Clang 23**, and the C status page updated.

FOLLOW-UP · #199658

```
static void g(void);
void f(void) { … extern void g(void); }
```

Same shadow-broken linkage inheritance, just for **functions** — it merges through `MergeFunctionDecl`, which this patch doesn't touch.

ISSUE 03 · #202945 · MONOREPO-WIDE

Getting `-Wunused-template` into `-Wall`

The warning has existed since 2017, but it ships `DefaultIgnore` and is commented out of the Unused group.

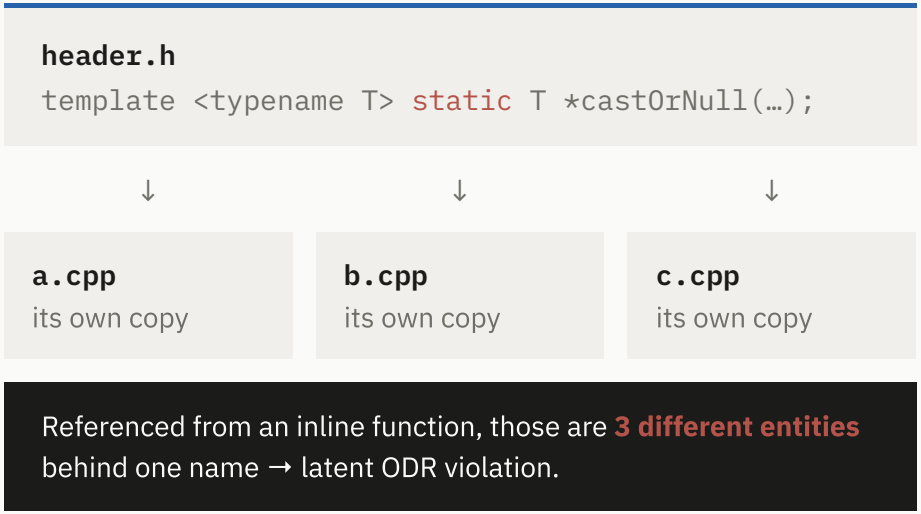
Step 1: clean every remaining occurrence across the LLVM monorepo. **Step 2:** turn it on, so the pattern can't come back.

ISSUE 03 · WHY IT'S WORTH DOING

Why it's worth doing

A `static` function template in a header has **internal linkage**, so every including TU gets its own copy.

When another inline function in a header references it, those copies are **different entities**. That's ill-formed, no diagnostic required.



Unused templates also cost real compile time, which makes this a genuine hygiene flag rather than a stylistic one.

ISSUE 03 · THE METHOD

Every fix fell into three patterns

P1 Header template, declared static

Drop the internal linkage

```
~~static~~ T *castOrNull(…)
```

The real ODR fix. One entity, vague linkage, zero bloat.

P2 Never instantiated in any .cpp

Delete it

```
// dead code — the warning
// is telling the truth
```

No use, no caller, no reason to keep it.

P3 Only used in assert / LLVM_DEBUG

Mark [[maybe_unused]]

```
[[maybe_unused]] static
bool isSorted(ArrayRef<T>)
```

Compiled out of release builds, so it's legitimately unused there.

ISSUE 03 · CLEANUP

Cleaning the projects tree

652 → **46** → **24** → **0**

raw warnings — unique sites — files — after cleanup

Ten NFC PRs, area by area: Object/ELF · Mips · JITLink/ORC · VPlan · IR · clang AST & frontend · FunctionAttrs · OpenMP frontend · Support/MC/ProfileData helpers

The tree built clean, so I opened the enable PR. I thought the cleanup was done. **It wasn't.**

ISSUE 03 · THE SECOND HALF

The half I'd never built

Projects

`llvm · clang · mlir · lld`

Built with the host clang. This is the half I had built and cleaned.

`0 warnings`

Runtimes

`compiler-rt · libc · openmp · flang-rt · offload`

Built with the just-built clang, so CI kept spitting warnings from code I had never compiled.

`where CI went red`

Full builds with the flag forced on locally, then GitHub Codespaces for the Linux-only parts my machine can't build: `hwasan`, `msan`, `flang-rt`, the GPU plugins.

20 merged NFC PRs across the whole campaign, and the warning is **live on main since 2026-07-08**.

MID-TERM STATUS

Where things stand

3

diagnostics: one merged, one in final review, one live on main

22

merged NFC PRs across the `-Wunused-template` campaign

C 23

N3410 conformance claimed for Clang 23; C status page updated

Also got commit access to llvm-project along the way...

PROCESS

How I used AI along the way

As a force multiplier, with the compiler as the judge.

---

01 The `clang:diagnostics` backlog is huge, so I had Claude sift it for confirmed, unowned bugs I could actually finish. That's how I found #46362, and there's a lot more sitting in there.

---

02 Trivial patches moved fast, but nothing went up without a real build and test run behind it, no matter how small the change looked.

---

03 I keep working notes distilled from standards drafts, reviews, and mailing lists. The C vs. C++ linkage answer came straight out of those.

SUMMARY

Three places Clang stayed quiet, all moving upstream.

Function-reference null comparisons `merged`

Mixed linkage in C `final review, Clang 23`

`-Wunused-template` in `-Wall` `live on main`

Aditya · flash1729 · C++ Alliance Fellowship