Subsetting and restricting C++ for memory safety

Subsetting

- Subsetting C++ is possible
- Subsetting meaningfully improves the safety profile (ha!) of C++
- Subsetting is practically adoptable in large production codebases
- Existence proof: WebKit
  - Adopted ~4MLoC of code (i.e. w/o comments, tests, external libs, etc)
  - Closes multiple classes of vulnerabilities, current policies would have prevented majority of historical exploits
  - Found (and prevented exploitability) of new and existing bugs

What is the point of these slides

- High level view of the possibility and effectiveness of subsetting
  - Including *existing* and *deployed* subset enforcement as a path for improving the security profile of the language.
- Not discussing P3589, P3700, P3716, or any others
  - People have opinions
  - I'm only going to talk about the *concept* of subsetting as a potential path forward.

P{3589,3700,3716} Profiles

- Not part of this presentation
  - People have opinions about P{3589,3700,3716} that are separate from “is subsetting possible or effective”
- I just want to talk about the *concept* of subsetting as a method to improve memory safety
  - Not discussing how it is specified
  - Not discussing syntax or anything

Profiles in C++ today

What is he talking about

- This information has been covered in various blog posts and presentations previously, most relevantly
  - C++ Memory Safety in WebKit (YouTube: C++Now 2025)
  - Recipe for Eliminating Entire Classes of Memory Safety Vulnerabilities in C and C++ (YouTube EuroLLVM 2025)
- Tangential: Bounds safety in C
  - “-fbounds-safety”: Enforcing bounds safety for production C code (YouTube EuroLLVM 2023)

What are we protecting today

Memory safety is a lot of things

- A subset that covers the major and most common flaws
  - Bounds safety
  - General lifetime safety
- Not being discussed here
  - Interior mutation and similar (people are working on this, but I don’t have anything concrete to say)
  - Concurrency

Requirements

- Local analysis where possible
  - Errors should cause local build failures
- Don’t require static analysis
  - Not 100% accurate
  - Not reasonable as part of standard developer work flow

General approach

- Subset the language
  - e.g profiles, but without language support
- Don’t try to verify safety of arbitrary code
  - Disallow operations that cannot be completely validated
  - Require correctness by construction

Bounds safety

- -Wunsafe-buffer-usage
  - Warning for unverifiable pointer operations: arithmetic, indexing, etc
- Forces use of safe APIs (spans, collections, etc)
- Bounds safety profile: `-Werror=unsafe-buffer-usage`
  - “unsafe” code regions: macros that selectively enable/disable the warning

Type safety

- More subsetting!
- Disallow all unsafe casts
  - Provide custom cast functions
  - Macros provide “unsafe” regions used to implement casting functions
- Also: endless allocator and codegen (e.g. pointer authentication) mitigations, but prefer correctness by construction

Lifetime safety

- Currently not addressing interior mutability
- Require RAII protection for types with non-trivial lifetimes
  - Cannot simply assume “correct” constructor or destructor behavior
  - Cannot make mutation assumptions
- Require trivially obvious lifetime protection
  - Don’t try to prove correctness
  - Require locally visible owner

Real world experience

WebKit vs bounds safety

- “Profile”: -Werror -Wunsafe-buffer-usage
- Only permitted pointer use is direct dereference (*ptr, ptr[0])
- Hardened standard library, including hardened iterators
- Incrementally adopted
  - 94% code coverage in less than a year
  - 0% performance cost (End-to-end)

Real world experience

WebKit vs type safety

- Custom cast functions (downcast<T>)
- Incrementally adopted
  - 98% adoption in less than a year
  - 0% performance cost
- Additional backstop: extensive allocator hardening, including type isolation

Real world experience

WebKit vs lifetime safety

- Caveat: Focused on reference counting
  - Most practical form of correct complex lifetime management
  - Already widely used throughout webkit
- Do not try to reason about lifetimes
  - Require local smart pointer (in WK: Ref, RefPtr, WeakPtr, …) owning any refcounted object
  - Or semantically guaranteed ownership (above argument means function arguments must be protected)

Real world experience

WebKit vs lifetime safety

- Incrementally adopted
- 90% coverage in less than a year
- 0*% Performance cost
  - Some code did need to be restructured

Real world experience

Implementation reality

- -Wunsafe-buffer-usage is already in the clang frontend
- C++ does not provide a mechanism to
  - Restrict cast expressions
  - Specify what objects need to be protected
  - Specify how they are protected
- Enforcement of cast and ownership rules via libAnalyzer
  - Local enforcement, but static analyzer is used because semantics cannot be specified in C++ directly

-fbounds-safety

- When std::span and similar aren’t an option (C, ABI constraints)
- Annotations to specify bounds rules for pointers
- No pointer indexing/arithmetic without bounds information
- Local pointer variables are wide pointers by default (e.g no annotations needed by default)
- Higher adoption cost than C++ options
- Adoption and deployment of -fbounds-safety has prevented multiple CVEs from being exploited

TLDR

- Subsetting C++ is a viable path to *improving* memory safety in C++
- Subsetting supports incremental adoption
- Subsetting allows meaningful and effective improvements in memory safety
  - WebKit's subsetting trivially break entire classes of exploits, have been shown to prevent the majority of historical attacks, and adoption has found bugs that previously would not have been detected.

Links

- C++ Memory Safety in WebKit - Geoffrey Garen - C++Now 2025
  - https://www.youtube.com/watch?v=RLw13wLM5Ko
  - https://schedule.cppnow.org/wp-content/uploads/2025/03/CPPNow-2025-C-Memory-Safety-in-WebKit.pdf
- 2025 EuroLLVM - Recipe for Eliminating Entire Classes of Memory Safety Vulnerabilities in C and C++
  - https://www.youtube.com/watch?v=rYOCPBUM1Hs
- 2023 EuroLLVM - Keynote: “-fbounds-safety”: Enforcing bounds safety for production C code
  - https://www.youtube.com/watch?v=RK9bfrsMdAM

Yet more links

- Hardening Techniques from the trenches (hardened stdlib)
  - https://www.youtube.com/watch?v=t7EJTO0-reg
- Clang bounds safety documentation: https://clang.llvm.org/docs/BoundsSafety.html
- “Fortify your app” apple WWDR
  - Bounds safety: https://youtu.be/UZeSyodAszc?t=13458
  - Should be split up at some point. You can also enjoy streaming of a schedule for the entire period of the lunch breaks, etc.