

Meeting C++ 2020

ONLINE

September 24

2020: The Year of Sanitizers?

Victor Ciura

Principal Engineer



@ciura_victor



CAPHYON

Abstract

Clang-tidy is the go-to assistant for most C++ programmers looking to improve their code, whether to modernize it or to find hidden bugs with its built-in checks. Static analysis is great, but you also get tons of false positives.

Now that you're hooked on smart tools, you have to try dynamic/runtime analysis. After years of improvements and successes for Clang and GCC users, LLVM AddressSanitizer (ASan) is finally available on Windows, in the latest Visual Studio 2019 versions. Let's find out how this experience is for MSVC projects.

We'll see how AddressSanitizer works behind the scenes (compiler and ASan runtime) and analyze the instrumentation impact, both in perf and memory footprint. We'll examine a handful of examples diagnosed by ASan and see how easy it is to read memory snapshots in Visual Studio, to pinpoint the failure.

Want to unleash the memory vulnerability beast? Put your test units on steroids, by spinning fuzzing jobs with ASan in Azure, leveraging the power of the Cloud from the comfort of your Visual Studio IDE.



Due to the nature of delivery medium & streaming delays (up to 15-20 sec), I prefer to take questions at the end*

Q & A



2020: The Year of Sanitizers?



Vignette in 3 parts

Static Analysis

Dynamic Analysis

Warm Fuzzy Feelings

Humans Depend on Tools



Programmers Depend on Tools

good code editor
(or IDE)

recent compiler(s)
[conformant/strict]

linter/formatter

perf profiler

powerful (visual) debugger

test framework

automated refactoring tools

build system

static analyzer

package manager

CI/CD service

dynamic analyzer
(runtime)

SCM client

code reviews platform

+ fuzzing

Why Do I Care ?

17 year old code base under active development
3.5 million lines of C++ code
a few brave nerds...

or

“How we manage to **clang-tidy** our whole code base,
while maintaining our monthly release cycle”

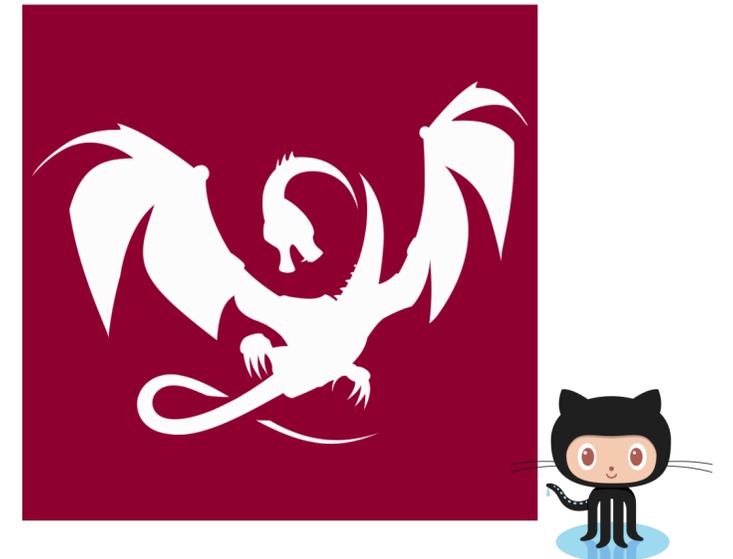
youtube.com/watch?v=Wl-9ozmxXbo

(CppCon 2017)

Who Am I?



Advanced Installer



Clang Power Tools

 **@ciura_victor**

Part I

Static Analysis



C++ Core Guidelines Checker



docs.microsoft.com/en-us/cpp/code-quality/quick-start-code-analysis-for-c-cpp

docs.microsoft.com/en-us/cpp/code-quality/code-analysis-for-cpp-corecheck



devblogs.microsoft.com/cppblog/new-safety-rules-in-c-core-check/

VS 16.7



Standard C/C++ rule sets

Visual Studio includes these standard sets of rules for native code:

Rule Set	Description
C++ Core Check Arithmetic Rules	These rules enforce checks related to arithmetic operations from the C++ Core Guidelines .
C++ Core Check Bounds Rules	These rules enforce the Bounds profile of the C++ Core Guidelines .
C++ Core Check Class Rules	These rules enforce checks related to classes from the C++ Core Guidelines .
C++ Core Check Concurrency Rules	These rules enforce checks related to concurrency from the C++ Core Guidelines .
C++ Core Check Const Rules	These rules enforce const-related checks from the C++ Core Guidelines .
C++ Core Check Declaration Rules	These rules enforce checks related to declarations from the C++ Core Guidelines .
C++ Core Check Enum Rules	These rules enforce enum-related checks from the C++ Core Guidelines .
C++ Core Check Experimental Rules	These rules collect some experimental checks. Eventually, we expect these checks to be moved to other rulesets or removed completely.
C++ Core Check Function Rules	These rules enforce checks related to functions from the C++ Core Guidelines .
C++ Core Check GSL Rules	These rules enforce checks related to the Guidelines Support Library from the C++ Core Guidelines .



docs.microsoft.com/en-us/cpp/code-quality/code-analysis-for-cpp-corecheck

ICYMI

Static Analysis

Visual Studio integrates with

- MSVC Code Analysis <https://aka.ms/cpp/ca/bg>
- Clang-tidy <https://aka.ms/cpp/clangtidy>
- Visual Studio Code Linters <https://aka.ms/cpp/linter>

✦ New C++ Core Checkers in MSVC Code Analysis

- Missing default label in switch statements
- Unannotated fall through in switch statements
- Expensive range-for copy
- Expensive copy with the auto keyword



Tue 9/15 12:00 – 13:00

Closing the Gap between Rust and C++ Using Principles of Static Analysis

Sunny Chatterjee – *destroy_n()* venue





clang-tidy

~ 300 checks

clang.llvm.org/extra/clang-tidy/checks/list.html



clang-tidy

- `modernize-use-nullptr`
- `modernize-loop-convert`
- `modernize-use-override`
- `readability-redundant-string-cstr`
- `modernize-use-emplace`
- `modernize-use-auto`
- `modernize-make-shared` & `modernize-make-unique`
- `modernize-use-equals-default` & `modernize-use-equals-delete`



clang-tidy

- `modernize-use-default-member-init`
- `readability-redundant-member-init`
- `modernize-pass-by-value`
- `modernize-return-braced-init-list`
- `modernize-use-using`
- `cppcoreguidelines-pro-type-member-init`
- `readability-redundant-string-init` & `misc-string-constructor`
- `misc-suspicious-string-compare` & `misc-string-compare`
- `misc-inefficient-algorithm`
- `cppcoreguidelines-*`



clang-tidy

- `abseil-string-find-startswith`
- `boost-use-to-string`
- `bugprone-string-constructor`
- `bugprone-string-integer-assignment`
- `bugprone-string-literal-with-embedded-nul`
- `bugprone-suspicious-string-compare`
- `modernize-raw-string-literal`
- `performance-faster-string-find`
- `performance-inefficient-string-concatenation`
- `readability-redundant-string-cstr`
- `readability-redundant-string-init`
- `readability-string-compare`

string checks

clang-tidy checks

Tidy Checks

Quick Search 🔍

bugprone-argument-comment	<input type="checkbox"/>	Off
bugprone-assert-side-effect	<input type="checkbox"/>	Off
bugprone-bool-pointer-implicit-conversion	<input type="checkbox"/>	Off
bugprone-branch-clone	<input type="checkbox"/>	Off
bugprone-copy-constructor-init	<input type="checkbox"/>	Off
bugprone-dangling-handle	<input checked="" type="checkbox"/>	On
bugprone-...	<input type="checkbox"/>	Off
bugprone-...	<input type="checkbox"/>	Off
bugprone-...	<input type="checkbox"/>	Off
bugprone-forwarding-reference-overload	<input type="checkbox"/>	Off
bugprone-inaccurate-erase	<input type="checkbox"/>	Off
bugprone-incorrect-roundings	<input type="checkbox"/>	Off
bugprone-integer-division	<input type="checkbox"/>	Off
bugprone-lambda-function-name	<input type="checkbox"/>	Off
bugprone-macro-parentheses	<input type="checkbox"/>	Off
bugprone-macro-repeated-side-effects	<input type="checkbox"/>	Off
bugprone-misplaced-operator-in-strlen-in-alloc	<input type="checkbox"/>	Off
bugprone-misplaced-widening-cast	<input type="checkbox"/>	Off

Default Checks

Detect dangling references in value handles like `std::experimental::string_view`. These dangling references can be a result of constructing handles from temporary values, where the temporary is destroyed soon after the handle is created.





clang-tidy bugprone-dangling-handle



Detect dangling references in value handles like `std::string_view`

These dangling references can be a result of constructing handles from **temporary** values, where the temporary is destroyed **soon** after the handle is created.

Options:



HandleClasses

A semicolon-separated list of class names that should be treated as handles. By default only `std::string_view` is considered.

<https://clang.llvm.org/extra/clang-tidy/checks/bugprone-dangling-handle.html>

Lifetime profile v1.0

Lifetime safety: Preventing common dangling

This is important because it turns out to be **easy** to convert **[by design]** a `std::string` to a `std::string_view`, or a `std::vector/array` to a `std::span`, so that **dangling is almost the default behavior**.



CppCoreGuidelines

<https://github.com/isocpp/CppCoreGuidelines/blob/master/docs/Lifetime.pdf>

Lifetime profile v1.0

Lifetime safety: Preventing common dangling

```
void example()  
{  
    std::string_view sv = std::string("dangling"); // A  
    std::cout << sv;  
}
```

clang **-Wlifetime**

Experimental



CppCoreGuidelines

<https://github.com/isocpp/CppCoreGuidelines/blob/master/docs/Lifetime.pdf>

Lifetime profile v1.0

Lifetime safety: Preventing common dangling

```
void example()
{
    std::string_view sv = std::string("dangling"); // A
    std::cout << sv; // ERROR (lifetime.3): 'sv' was invalidated when
} // temporary was destroyed (line A)
```

clang **-Wlifetime**

Experimental



CppCoreGuidelines

<https://github.com/isocpp/CppCoreGuidelines/blob/master/docs/Lifetime.pdf>

Lifetime safety: Preventing common dangling

`[-Wdangling-gsl]` diagnosed by default in **Clang 10**

warning: initializing pointer member to point to a temporary object whose lifetime is shorter than the lifetime of the constructed object

```
void example()
{
    std::string_view sv = std::string("dangling");

    std::cout << sv;
}
```

<https://clang.llvm.org/docs/DiagnosticsReference.html#wdangling-gsl>

Lifetime safety: Preventing common dangling

`[-Wdangling-gsl]` diagnosed by default in **Clang 10**

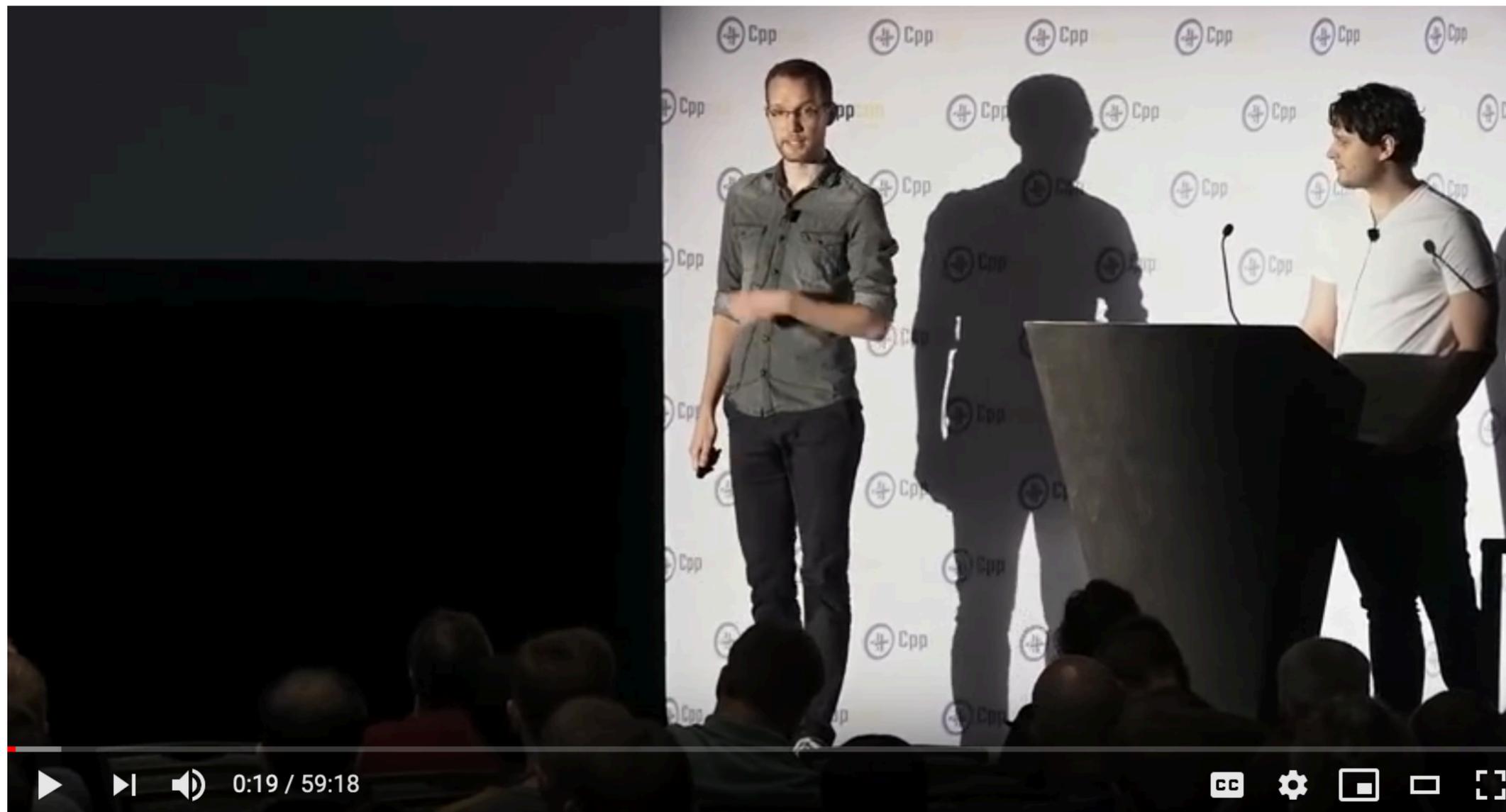
warning: initializing pointer member to point to a temporary object whose lifetime is shorter than the lifetime of the constructed object

```
void example()
{
    std::string_view sv = std::string("dangling");
        // warning: object backing the pointer will be destroyed
        // at the end of the full-expression [-Wdangling-gsl]
    std::cout << sv;
}
```

<https://clang.llvm.org/docs/DiagnosticsReference.html#wdangling-gsl>

Lifetime profile

<https://github.com/isocpp/CppCoreGuidelines/blob/master/docs/Lifetime.pdf>



 AURORA

CppCon 2019: Gábor Horváth, Matthias Gehre "Lifetime analysis for everyone"

<https://www.youtube.com/watch?v=d67kfSnhbpA>



clang-tidy

Checks are organized in **modules**, which can be linked into clang-tidy with minimal or no code changes in clang-tidy



clang-tidy

Checks are organized in **modules**, which can be linked into clang-tidy with minimal or no code changes in clang-tidy

Checks can plug into the analysis on the **preprocessor** level using **PPCallbacks** or on the **AST** level using **AST Matchers**



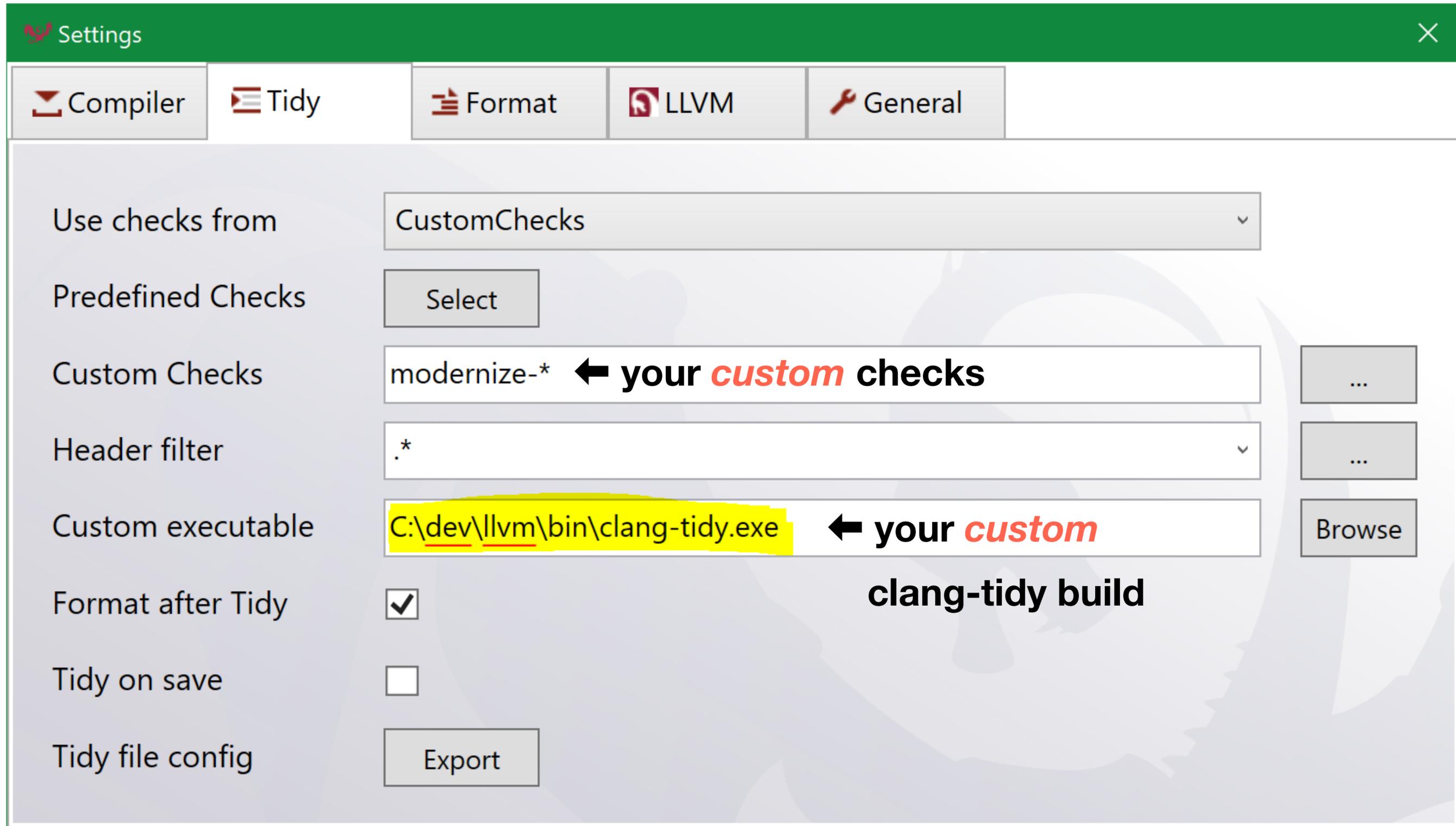
clang-tidy

Checks are organized in **modules**, which can be linked into clang-tidy with minimal or no code changes in clang-tidy

Checks can plug into the analysis on the **preprocessor** level using **PPCallbacks** or on the AST level using **AST Matchers**

Checks can **report** issues in a similar way to how Clang diagnostics work. A **fix-it** hint can be attached to a diagnostic message

Custom clang-tidy checks



The screenshot shows the 'Settings' dialog box in Visual Studio, specifically the 'Tidy' tab. The 'Use checks from' dropdown is set to 'CustomChecks'. The 'Predefined Checks' section has a 'Select' button. The 'Custom Checks' text box contains 'modernize-*' with an arrow pointing to it and the text '← your *custom* checks'. The 'Header filter' dropdown is set to '*.*. The 'Custom executable' text box contains 'C:\dev\llvm\bin\clang-tidy.exe' with a yellow highlight and an arrow pointing to it and the text '← your *custom* clang-tidy build'. The 'Format after Tidy' checkbox is checked. The 'Tidy on save' checkbox is unchecked. The 'Tidy file config' section has an 'Export' button.

Settings

Compiler Tidy Format LLVM General

Use checks from CustomChecks

Predefined Checks Select

Custom Checks modernize-* ← your *custom* checks

Header filter *.*

Custom executable C:\dev\llvm\bin\clang-tidy.exe ← your *custom* clang-tidy build

Format after Tidy

Tidy on save

Tidy file config Export

**Write *custom* checks for your needs
(project specific)**

Run them regularly !

Explore Further



code::dive 2018

Refactor with Clang Tooling

Tools, Tips, Tricks and Traps

Stephen Kelly
steveire.wordpress.com
@steveire

Stephen Kelly

<https://steveire.wordpress.com/2019/01/02/refactor-with-clang-tooling-at-codedive-2018/>

Explore Further

Cppcon | 2019
The C++ Conference | cppcon.org



`#include <C++>
#include <C++>
#include <C++>
#include <C++>
#include <C++>
#include <C++>
#include <C++>`

Fred Tingaud

Clang Based Refactoring

How to refactor millions of lines of code without alienating your colleagues

Fred Tingaud Murex @FredTingaudDev

Clang-based Refactoring,
How to refactor millions
of line of code without
alienating your colleagues

2

<https://www.youtube.com/watch?v=JPnN2c2odNY>

What About Developer Workflow?



+



2019 Victor Ciura | @ciura_victor

15



VICTOR CIURA

▶ | 🔊 17:09 / 1:00:34

CC HD 📺 📱 🖥️

📍 KINO | NOWE HORYZONTY

Status quo: clang-tidy & AddressSanitizer on Windows - Victor Ciura - code::dive 2019

Up next

AUTOPLAY

C++ Weekly - Ep 3 Intro to

www.youtube.com/watch?v=Iz4C29yul2U

2020 Victor Ciura | @ciura_victor - 2020: The Year of Sanitizers?

28



Explore Further

A new series of blog articles on [Visual C++ Team blog](#) by [Stephen Kelly](#)

Exploring Clang Tooling, Part 0: Building Your Code with Clang

<https://blogs.msdn.microsoft.com/vcblog/2018/09/18/exploring-clang-tooling-part-0-building-your-code-with-clang/>

Exploring Clang Tooling, Part 1: Extending Clang-Tidy

<https://blogs.msdn.microsoft.com/vcblog/2018/10/19/exploring-clang-tooling-part-1-extending-clang-tidy/>

Exploring Clang Tooling, Part 2: Examining the Clang AST with clang-query

<https://blogs.msdn.microsoft.com/vcblog/2018/10/23/exploring-clang-tooling-part-2-examining-the-clang-ast-with-clang-query/>



Explore Further

A new series of blog articles on [Visual C++ Team blog](#) by [Stephen Kelly](#)

Exploring Clang Tooling, Part 3: Rewriting Code with clang-tidy

<https://blogs.msdn.microsoft.com/vcblog/2018/11/06/exploring-clang-tooling-part-3-rewriting-code-with-clang-tidy/>

Exploring Clang Tooling: Using Build Tools with clang-tidy

<https://blogs.msdn.microsoft.com/vcblog/2018/11/27/exploring-clang-tooling-using-build-tools-with-clang-tidy/>



Explore Further

More blog articles by [Stephen Kelly](#)

Future Developments in clang-query

<https://steveire.wordpress.com/2018/11/11/future-developments-in-clang-query/>

Composing AST Matchers in clang-tidy

<https://steveire.wordpress.com/2018/11/20/composing-ast-matchers-in-clang-tidy/>

Visual Studio 2019

v16.2

Clang/LLVM support for MSBuild & CMake Projects

Ships with Clang (as optional component)

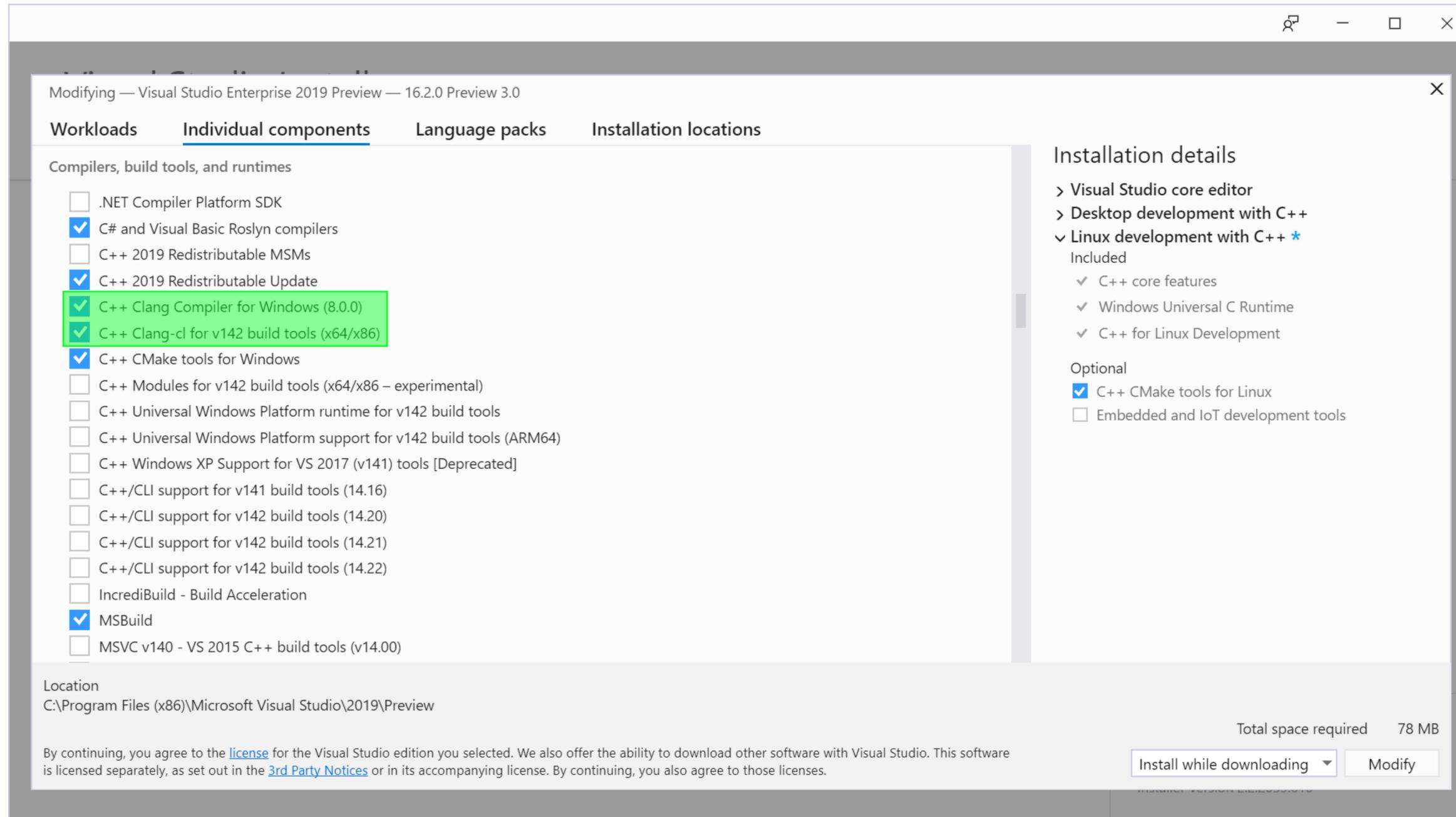
clang-cl.exe



<https://devblogs.microsoft.com/cppblog/clang-llvm-support-for-msbuild-projects/>

Visual Studio 2019

v16.2



Visual Studio 2019

v16.7

Modifying — Visual Studio Professional 2019 — 16.7.2

Workloads Individual components Language packs Installation locations

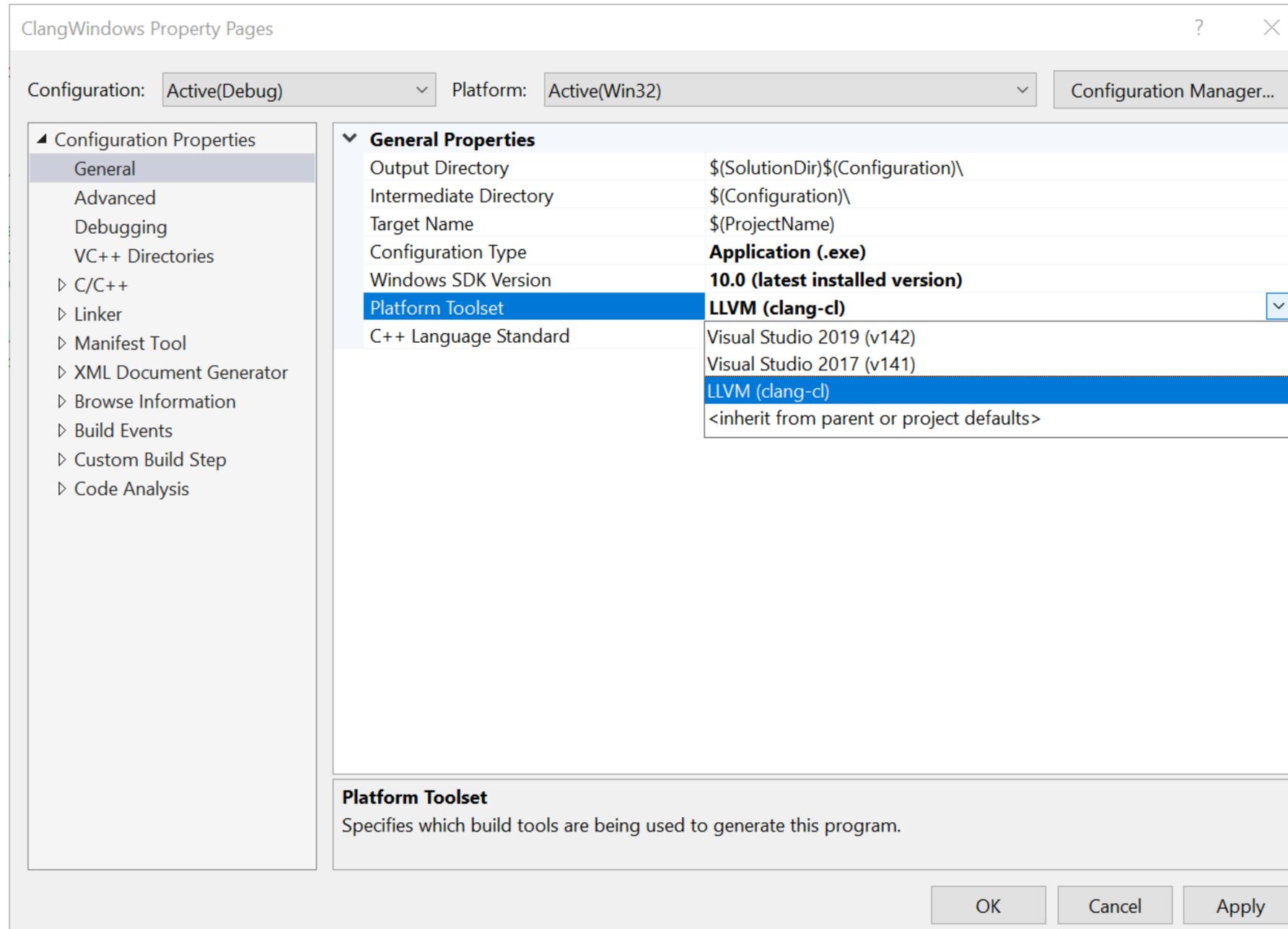
clang ×

Compilers, build tools, and runtimes

- C++ Clang Compiler for Windows (10.0.0) 
- C++ Clang-cl for v142 build tools (x64/x86)

Visual Studio 2019

v16.2



clang-cl.exe

Visual Studio 2019

v16.4

clang-tidy

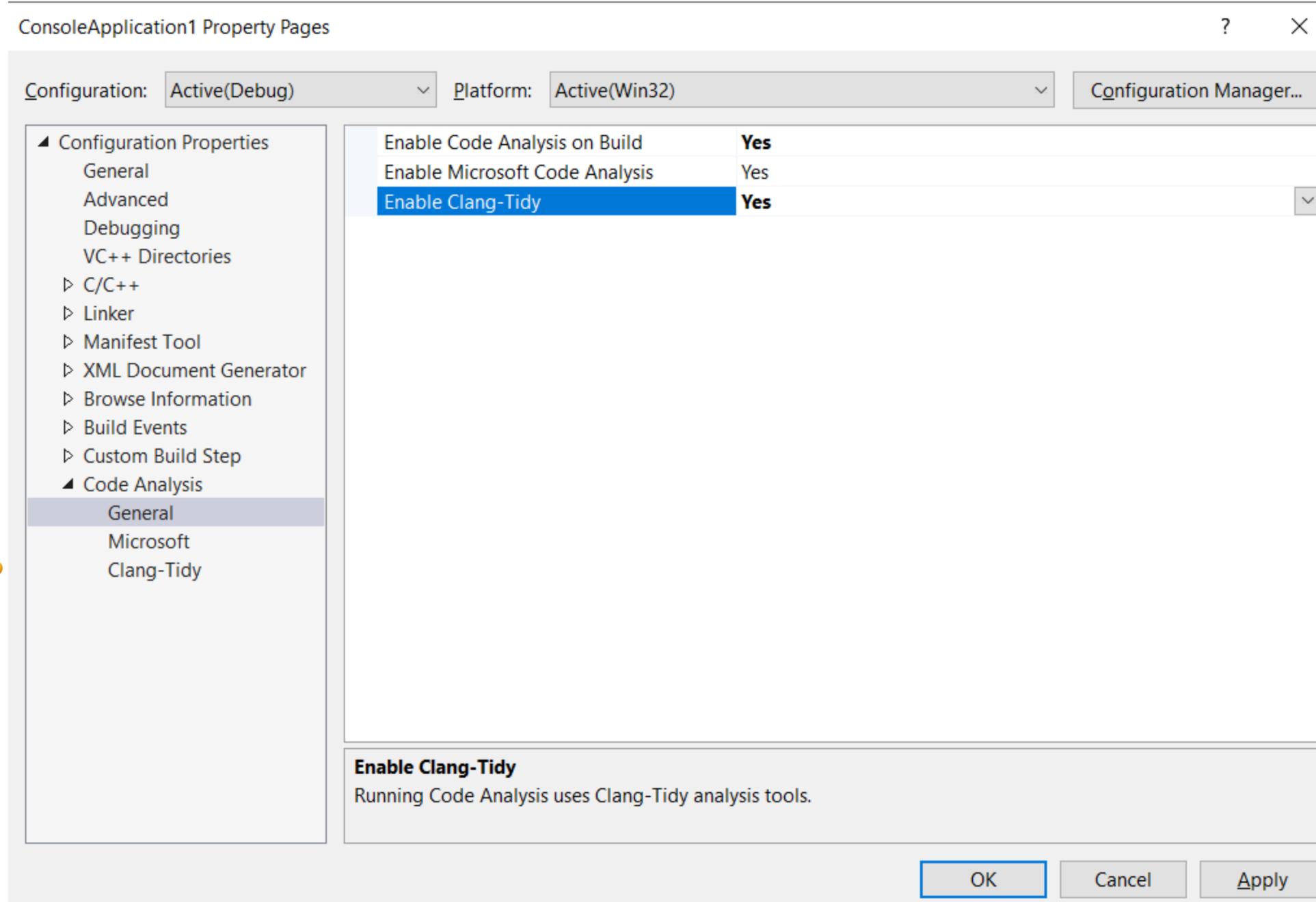
code analysis



<https://devblogs.microsoft.com/cppblog/code-analysis-with-clang-tidy-in-visual-studio/>

Visual Studio 2019

v16.4



ConsoleApplication1 Property Pages

Configuration: Active(Debug) Platform: Active(Win32) Configuration Manager...

- Configuration Properties
 - General
 - Advanced
 - Debugging
 - VC++ Directories
 - C/C++
 - Linker
 - Manifest Tool
 - XML Document Generator
 - Browse Information
 - Build Events
 - Custom Build Step
 - Code Analysis
 - General
 - Microsoft
 - Clang-Tidy

Enable Code Analysis on Build	Yes
Enable Microsoft Code Analysis	Yes
Enable Clang-Tidy	Yes

Enable Clang-Tidy
Running Code Analysis uses Clang-Tidy analysis tools.

OK Cancel Apply

Visual Studio 2019

v16.4

clang-tidy warnings

Error List

Entire Solution | 0 Errors | 10 Warnings | 0 Messages | Build + IntelliSense

Code	Description	File	Line	Col	Category
! readability-isolate-declaration	multiple declarations in a single statement reduces readability	CMAKEDEMO.CPP	23	2	readability
! modernize-use-nullptr	use nullptr	CMAKEDEMO.CPP	31	7	modernize
! cppcoreguidelines-macro-usage	macro 'TRUE' used to declare a constant; consider using a 'constexpr' constant	CMAKEDEMO.CPP	35	9	cppcoreguidelines
! clang-diagnostic-unused-variable	unused variable 'local'	CMAKEDEMO.CPP	50	13	clang-diagnostic
! clang-diagnostic-unused-const-variable	unused variable 'pos_x'	CMAKEDEMO.CPP	36	11	clang-diagnostic
! clang-diagnostic-uninitialized	variable 'numLives' is uninitialized when used here	CMAKEDEMO.CPP	24	3	clang-diagnostic
! clang-diagnostic-return-type	control reaches end of non-void function	CMAKEDEMO.CPP	32	1	clang-diagnostic
! clang-analyzer-core.NullDereference	Dereference of undefined pointer value	CMAKEDEMO.CPP	24	12	clang-analyzer

Error List | Output



<https://devblogs.microsoft.com/cppblog/code-analysis-with-clang-tidy-in-visual-studio/>

Visual Studio 2019

v16.4

clang-tidy warnings also display as in-editor squiggles

```
const int pos_x = 47;
```

```
enum Positio  
void tux(Pos
```

```
struct node
```

 const int pos_x = 47

Search Online

clang-diagnostic-unused-const-variable: unused variable 'pos_x'

Code Analysis runs automatically in the background

NOT on
Visual Studio 2019 v16.4+
yet ?

No problem



=



->



Clang Power Tools

www.clangpowertools.com

LLVM

clang-tidy

clang++

clang-format

clang-check/query

Visual Studio

2015 / 2017 / 2019

Static vs Dynamic Analysis

Static Analysis

- **offline** (out of the normal compilation cycle) => can take longer to process source code
- is intimately linked to the used **programming language**
- can detect a lot of **semantic issues**
- can yield a lot of **false positive** results (sometimes you go on a wild goose chase)
- very poor at **whole program analysis** (follow connections in different TUs)
- almost helpless around **virtual functions** (difficult to **de-virtualize** calls)
- weak analysis ability around **global pointers**
- **pointer aliasing** makes it hard to prove things (alias analysis is hard problem)
- vicious cycle: **type propagation** <> **alias analysis**

Dynamic Analysis

- sometimes **intrusive**: you need to compile the program in a special mode
- runtime overhead (**performance impact**: depending on tool, from **2x** up to **10x**)
- **extra-memory** usage (for memory related tools/instrumentation), 2x or more
- sometimes difficult to map error reports into **source code** for Release/**optimized builds** (symbols info, line numbers, inlined functions)
- some tools require to **recompile** the **whole program** in instrumented mode
- must integrate runtime analysis with **Test Units**
- must ensure good **code coverage** for the runtime analysis (all possible scenarios)
- the biggest impact when combined with **fuzzing**

Dynamic Analysis

- sometimes **intrusive**: you need to compile the program in a special mode
- runtime overhead (**performance impact**: depending on tool, from **2x** up to **10x**)
- **extra-memory** usage (for memory related tools/instrumentation), 2x or more
- sometimes difficult to map error reports into **source code** for Release/**optimized builds** (symbols info, line numbers, inlined functions)
- some tools require to **recompile** the **whole program** in instrumented mode
- must integrate runtime analysis with **Test Units**
- must ensure good **code coverage** for the runtime analysis (all possible scenarios)
- the biggest impact when combined with **fuzzing**

0 false positives!

Part II

Dynamic Analysis

ICYMI

Control Flow Guard

`/guard:cf`

Enforce control flow integrity (Windows 8.1 & Windows 10)

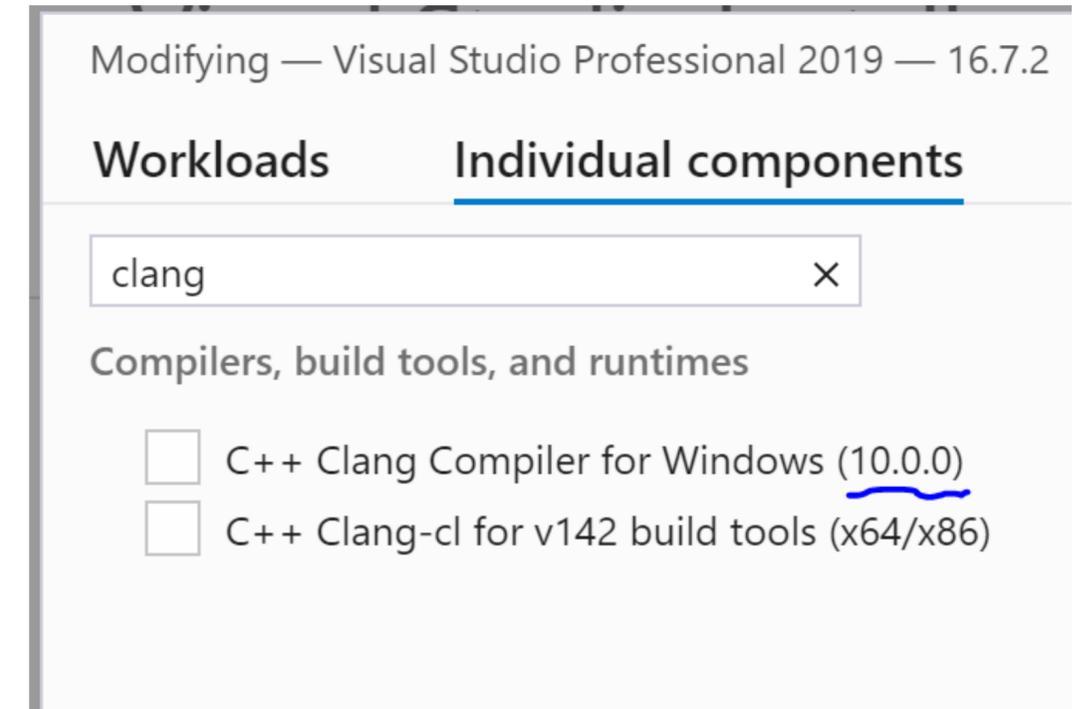
CFG is complementary to other exploit mitigations, such as:

- Address Space Layout Randomization (**ASLR**)
- Data Execution Prevention (**DEP**)

MSVC

CFG is now supported in **LLVM 10**

C++ & Rust



<https://aka.ms/cpp/cfg-llvm>

Sanitizers





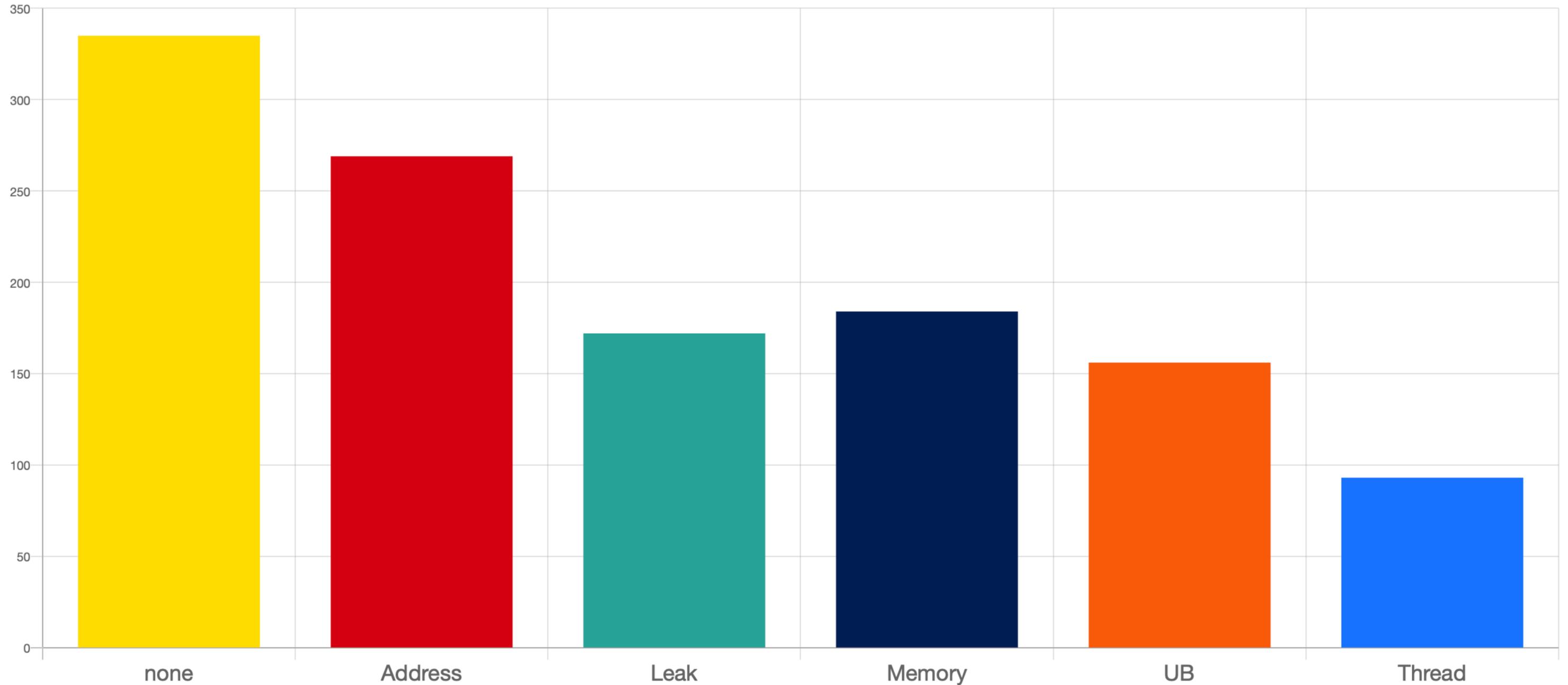
Sanitizers

- **AddressSanitizer** - detects addressability issues
- **LeakSanitizer** - detects memory leaks
- **ThreadSanitizer** - detects data races and deadlocks
- **MemorySanitizer** - detects use of uninitialized memory
- **HWASAN** - hardware-assisted AddressSanitizer (consumes less memory)
- **UBSan** - detects Undefined Behavior

github.com/google/sanitizers

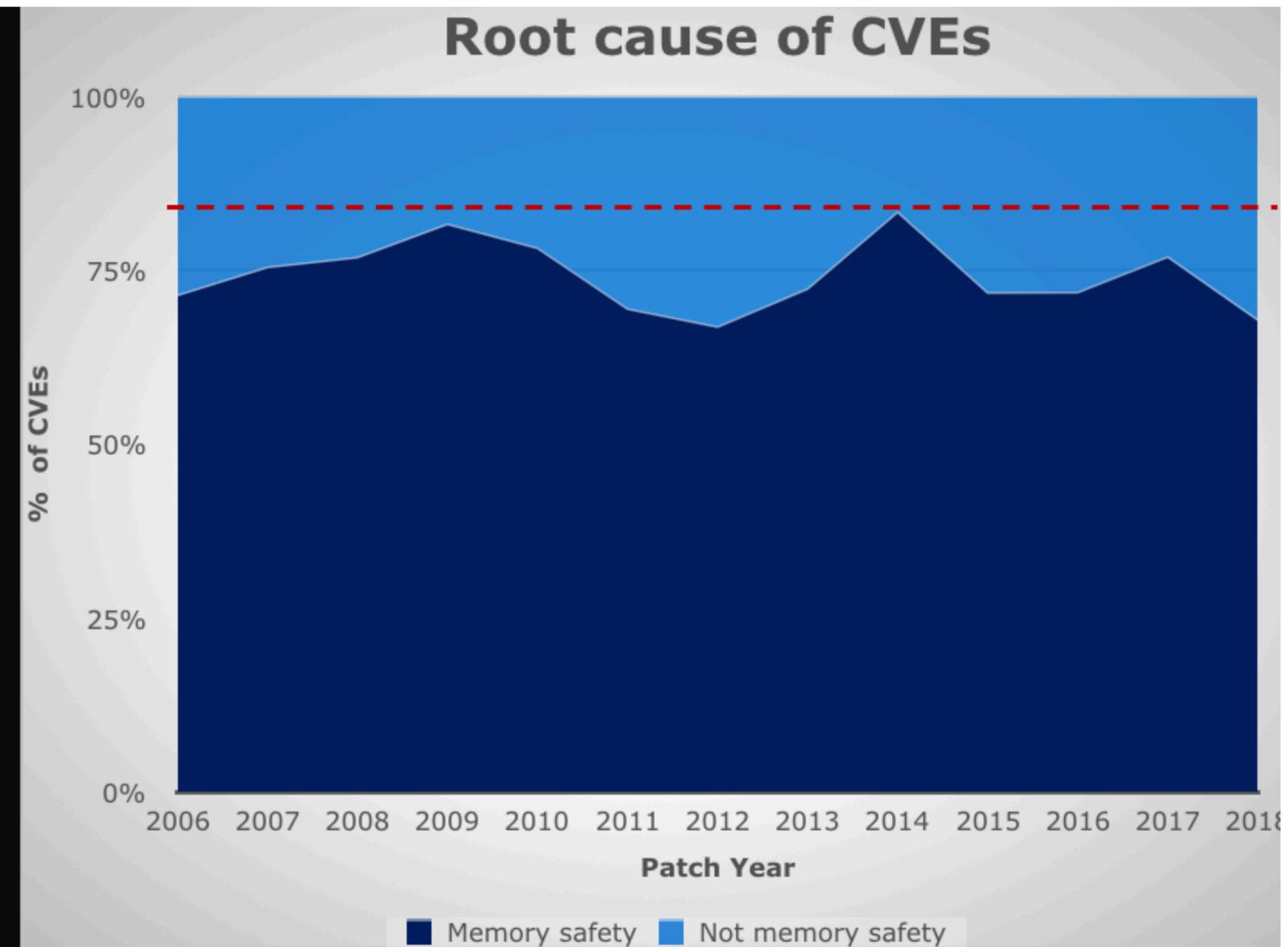
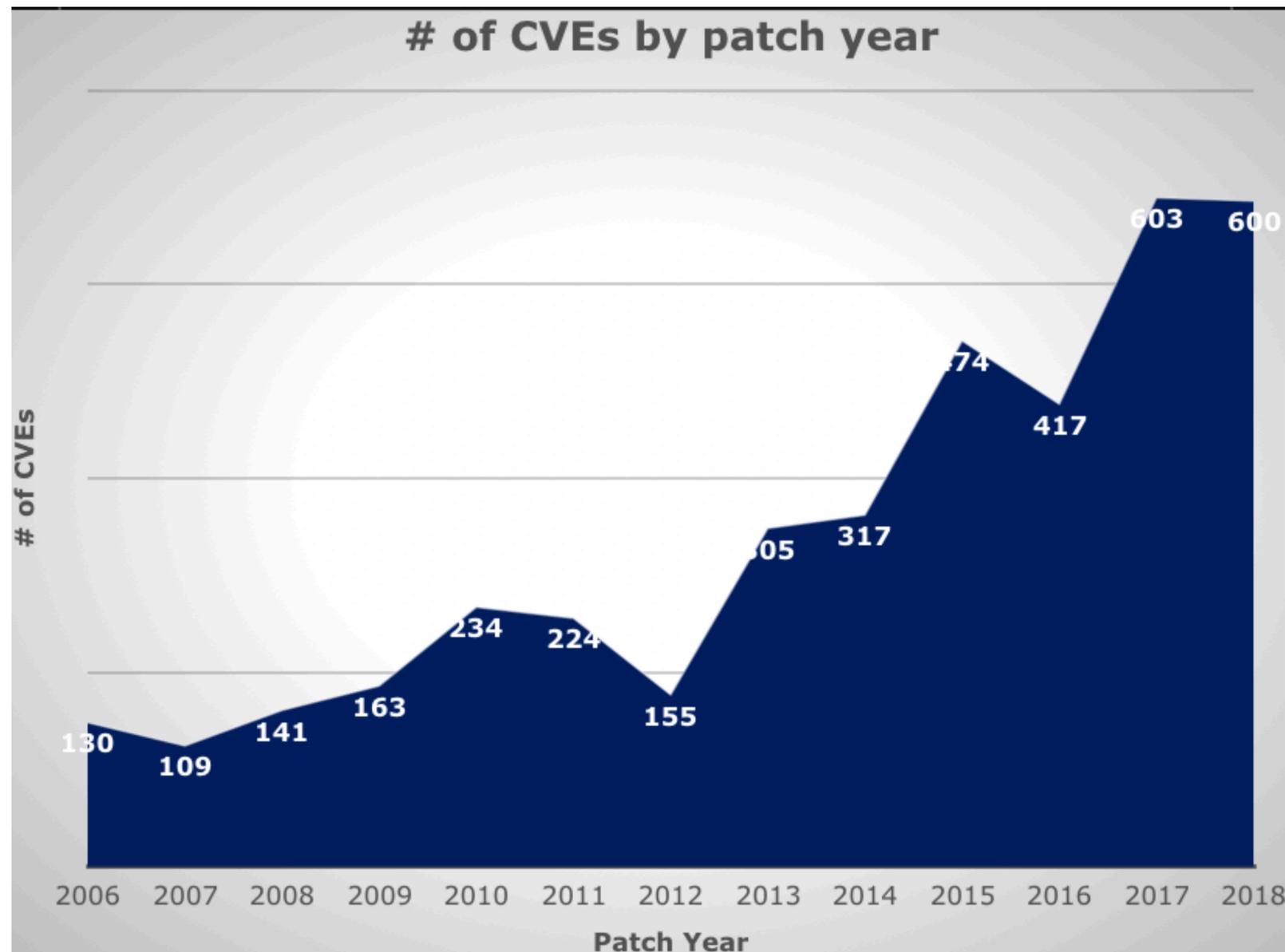
[Next Question](#) | [Survey results](#)

Meeting C++ Community Survey Which sanitizers do you use in your builds?



Common Vulnerabilities and Exposures

Memory safety continues to dominate



[youtube.com/watch?v=0EsqxGgYOQU](https://www.youtube.com/watch?v=0EsqxGgYOQU)



Address Sanitizer (ASan)

De facto standard for detecting **memory safety issues**

It's important for basic **correctness and sometimes true **vulnerabilities****

github.com/google/sanitizers/wiki/AddressSanitizer



Address Sanitizer (ASan)

Detects:

- **Use after free** (dangling pointer dereference)
- **Heap buffer overflow**
- **Stack buffer overflow**
- **Global buffer overflow**
- **Use after return**
- **Use after scope**
- **Initialization order bugs**
- **Memory leaks**

github.com/google/sanitizers/wiki/AddressSanitizer



Address Sanitizer (ASan)

Started in **LLVM** by a team @ Google
and quickly took off as a *de facto* industry standard
for runtime program analysis

github.com/google/sanitizers/wiki/AddressSanitizer



Address Sanitizer (ASan)

LLVM starting with version **3.1** (2012)

GCC starting with version **4.8** (2013)

MSVC starting with VS **16.4** (late 2019)

Visual Studio 2019

v16.4

October 2019

Address Sanitizer (ASan)



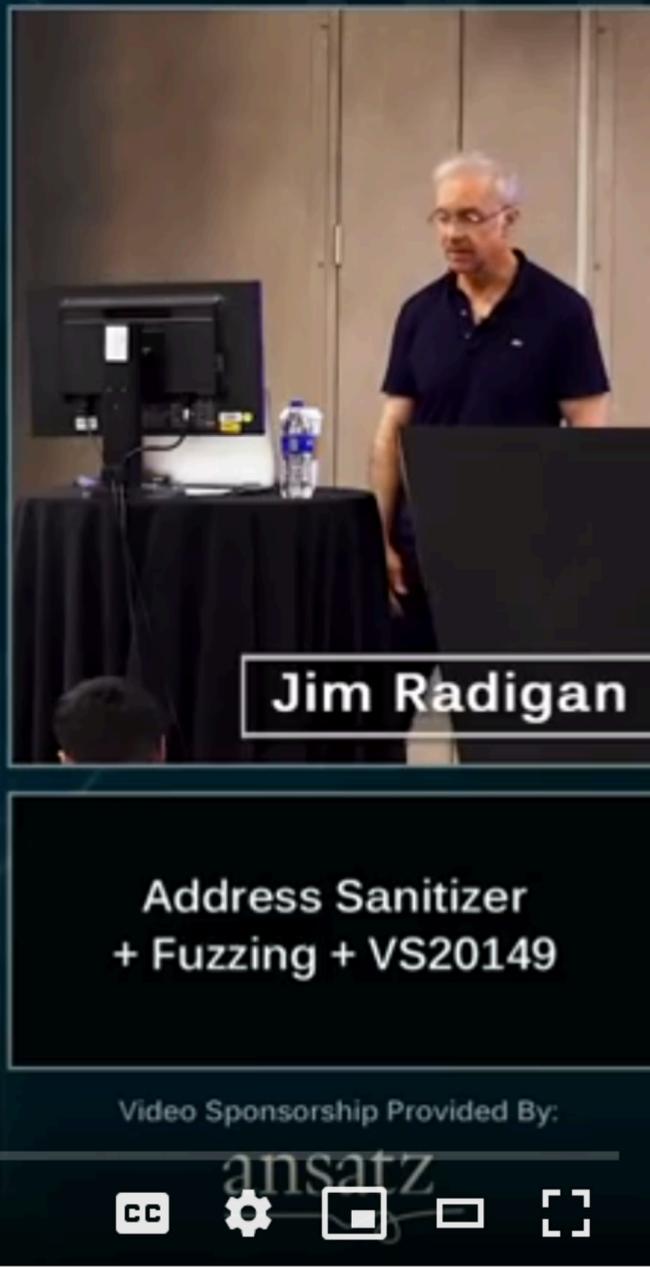
devblogs.microsoft.com/cppblog/addresssanitizer-asan-for-windows-with-msvc/



Address Sanitizer + Fuzzing + VS2019

jradigan@Microsoft.com

Visual Studio 2019 launch



Jim Radigan

Address Sanitizer + Fuzzing + VS20149

Video Sponsorship Provided By: ansatz

AURORA
CppCon 2019: Jim Radigan C++ Sanitizers and Fuzzing for the Windows Platform Using New Compilers...

<https://www.youtube.com/watch?v=0EsqxGgYOQU>

Visual Studio 2019

v16.4

Modifying — Visual Studio Enterprise 2019 Int Preview — 16.4.0 Preview 3.0 [29408.177.master]

Workloads Individual components Language packs Installation locations

Web & Cloud (4)

- ASP.NET and web development
Build web applications using ASP.NET Core, ASP.NET, HTML/JavaScript, and Containers including Docker support.
- Python development
Editing, debugging, interactive development and source control for Python.
- Azure development
Azure SDKs, tools, and projects for developing cloud apps and creating resources using .NET Core and .NET...
- Node.js development
Build scalable network applications using Node.js, an asynchronous event-driven JavaScript runtime.

Windows (3)

- .NET desktop development
Build WPF, Windows Forms, and console applications using C#, Visual Basic, and F# with .NET Core and .NET...
- Desktop development with C++
Build modern C++ apps for Windows using tools of your choice, including MSVC, Clang, CMake, or MSBuild.
- Universal Windows Platform development
Create applications for the Universal Windows Platform with C#, VB, or optionally C++.

Installation details

✓ Desktop development with C++
Included

- ✓ C++ core desktop features
- ✓ IntelliCode

Optional

- MSVC v142 - VS 2019 C++ x64/x86 build tools (...)
- Windows 10 SDK (10.0.18362.0)
- Just-In-Time debugger
- C++ profiling tools
- C++ CMake tools for Windows
- C++ ATL for latest v142 build tools (x86 & x64)
- Test Adapter for Boost.Test
- Test Adapter for Google Test
- Live Share
- C++ AddressSanitizer (Experimental)
- IntelliTrace
- C++ MFC for latest v142 build tools (x86 & x64)
- C++/CLI support for v142 build tools (14.24)
- C++ Modules for v142 build tools (x64/x86 - ex...
- C++ Clang tools for Windows (8.0.1 - x64/x86)

Location
C:\Program Files (x86)\Microsoft Visual Studio\2019\Preview

Total space required 0 KB

By continuing, you agree to the [license](#) for the Visual Studio edition you selected. We also offer the ability to download other software with Visual Studio. This software is licensed separately, as set out in the [3rd Party Notices](#) or in its accompanying license. By continuing, you also agree to those licenses.

Install while downloading



Visual Studio 2019

v16.4

Modifying — Visual Studio Professional 2019 — 16.7.2

Workloads Individual components Language packs Installation locations

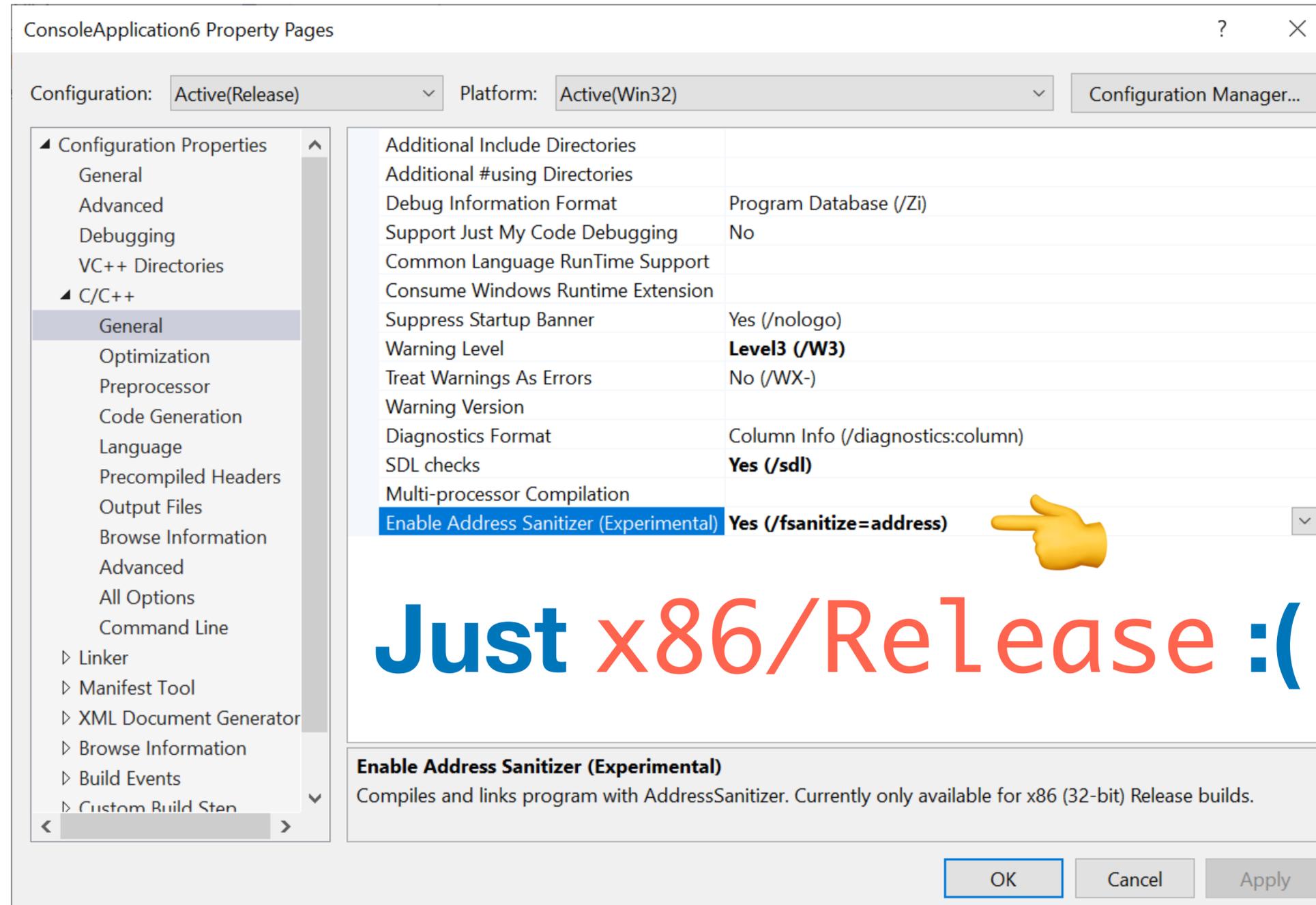
sanitizer × 

Debugging and testing

C++ AddressSanitizer (Experimental)

C++ AddressSanitizer (Experimental)

AddressSanitizer (ASAN) is a tool for detecting memory errors in C/C++ code. ASAN uses instrumentation to check memory accesses and report any memory safety issues. This feature is experimental and should not be used outside of testing environments



October 2019

Visual Studio 2019

v16.7

SystemScanner Property Pages

Configuration: Debug Platform: All Platforms Configuration Manager...

- Configuration Properties
 - General
 - Advanced
 - Debugging
 - VC++ Directories
- C/C++
 - General**
 - Optimization
 - Preprocessor
 - Code Generation
 - Language
 - Precompiled Headers
 - Output Files
 - Browse Information
 - Advanced
 - All Options
 - Command Line
- Librarian
- XML Document Generator
- Browse Information
- Build Events
- Custom Build Step
- Code Analysis

Additional Include Directories	<code>\$(ProjectDir);..\..\.;%(AdditionalIncludeDirectories)</code>
Additional #using Directories	
Debug Information Format	Program Database (/Zi)
Support Just My Code Debugging	No
Common Language RunTime Sup	
Consume Windows Runtime Exter	
Suppress Startup Banner	Yes (/nologo)
Warning Level	Level4 (/W4)
Treat Warnings As Errors	Yes (/WX)
Warning Version	
Diagnostics Format	Caret (/diagnostics:caret)
SDL checks	
Multi-processor Compilation	Yes (/MP)
Enable Address Sanitizer (Experimental)	Yes (/fsanitize=address)

x64 & Debug builds

Enable Address Sanitizer (Experimental)
Compiles and links program with AddressSanitizer. Currently available for x86 and x64 builds.

OK Cancel Apply



August 2020



August 2020

Visual Studio 2019

v16.7

+ x64 & Debug builds

support all Debug runtimes: /MTd /MDd

docs.microsoft.com/en-us/visualstudio/releases/2019/release-notes#16.7.0

Visual Studio 2019

v16.7

ASan features:

- stack-use-after-scope
- stack-buffer-overflow
- stack-buffer-underflow
- heap-buffer-overflow (no underflow)
- heap-use-after-free
- calloc-overflow
- dynamic-stack-buffer-overflow (alloca)
- global-overflow (C++ source code)
- new-delete-type-mismatch
- memcpy-param-overlap
- allocation-size-too-big
- invalid-aligned-alloc-alignment
- use-after-poison
- intra-object-overflow
- initialization-order-fiasco
- double-free
- alloc-dealloc-mismatch



Soon...

Visual Studio 2019

v16.8

ASan features:

- `global 'C' variables`
(in C a global can be declared many times, and each declaration can be of a different type and size)
- `__declspec(no_sanitize_address)`
(opt out of instrumenting entire functions or specific variables)
- `automatically link appropriate ASan libs`

Future versions:

- `use-after-return (opt-in)`
(requires code gen that utilizes two stack frames for each function)



September 14

Visual Studio 2019 v16.8 Preview 3

Cppcon The C++ Conference

Hallway Track AMAs Help Desk[™] CppCon.org[™] Slack[™] Sched[™] Expo Hall

all_of() basics count_if() destroy_n() embedded fuzzing generate_n()

count_if() Track
Break out session track at CppCon 2020

LIVE 84

Microsoft

A New Decade of
Visual Studio
C++20, Open STL, and More

Marian Luparu @mluparu
Sy Brand @TartanLlama

C++ Product Team, Microsoft
@VisualC
<https://aka.ms/cpp>

CppCon 2020

Chat Participants Q&A

General Chat

Marian Luparu
Folks, we're starting in 5 minutes
7:27 PM | Today

Marian Luparu
Oh, make that 3 :)
7:27 PM | Today

Brandon David Powers (He/Him)
hype
7:28 PM | Today

Type a message

Need help?

Raise hand Quit Event

devblogs.microsoft.com/cppblog/a-multitude-of-updates-in-visual-studio-2019-version-16-8-preview-3/

Visual Studio ASan

Experimental



Help needed: Report bugs!

Very soon out of Experimental

Visual Studio ASan Experimental

Very tall order to bring ASAN to **Windows**



Challenges bringing ASan to Windows

the surface area of the Microsoft platform is enormous

Challenges bringing ASan to Windows

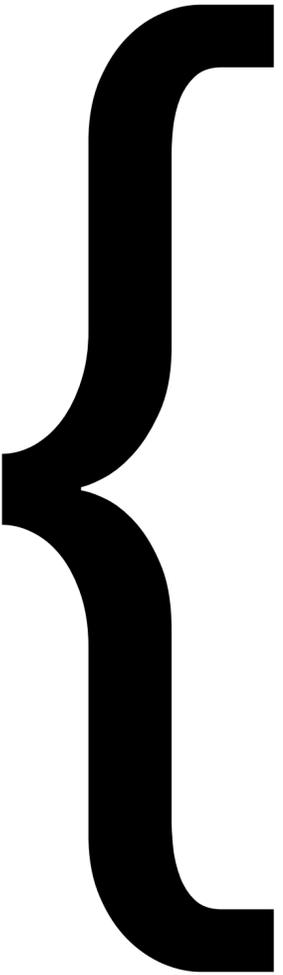
the surface area of the Microsoft platform is enormous

non-standard C++

Challenges bringing ASan to Windows

the surface area of the Microsoft platform is enormous

non-standard C++

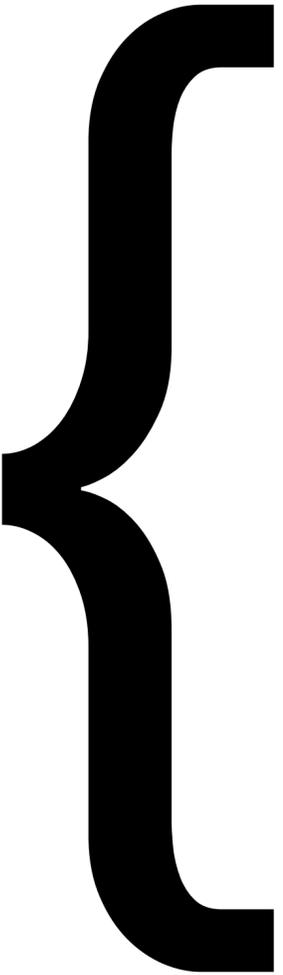


Challenges bringing ASan to Windows

the surface area of the Microsoft platform is enormous

Structured Exception Handling (SEH) /EHa

non-standard C++



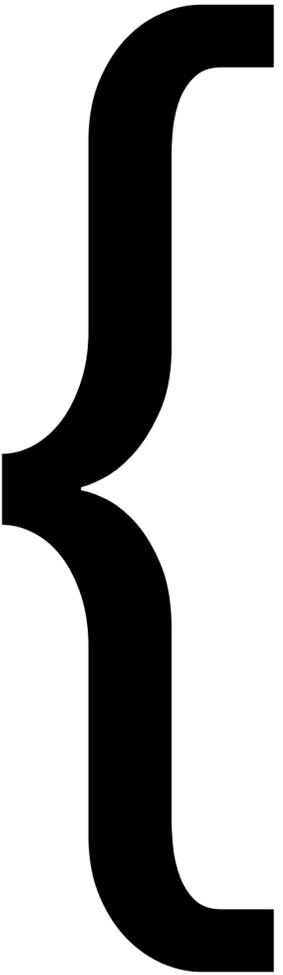
Challenges bringing ASan to Windows

the surface area of the Microsoft platform is enormous

Structured Exception Handling (SEH) `/EHa`

AV traps `0xc0000005`

non-standard C++



Challenges bringing ASan to Windows

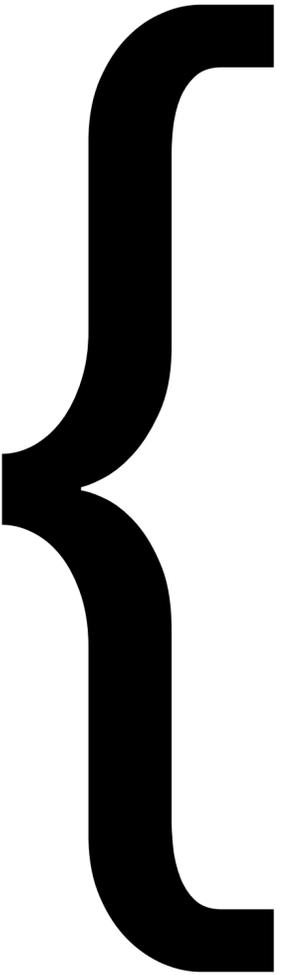
the surface area of the Microsoft platform is enormous

Structured Exception Handling (SEH) `/EHa`

AV traps `0xc0000005`

vast amount of legacy code (really, really, really OLD code)

non-standard C++



Challenges bringing ASan to Windows

the surface area of the Microsoft platform is enormous

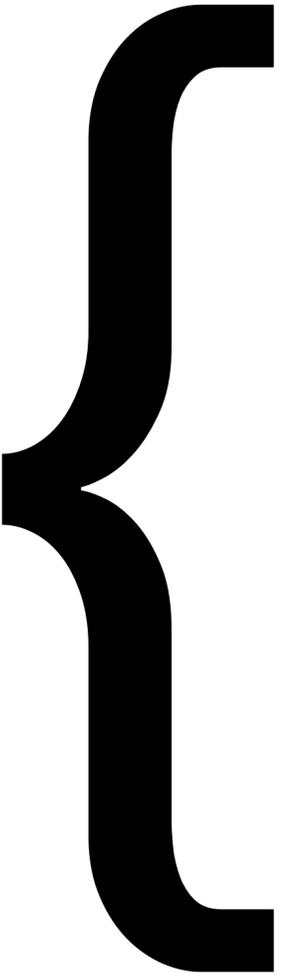
Structured Exception Handling (SEH) `/EHa`

AV traps `0xc0000005`

vast amount of legacy code (really, really, really OLD code)

COM

non-standard C++



Challenges bringing ASan to Windows

the surface area of the Microsoft platform is enormous

Structured Exception Handling (SEH) `/EHa`

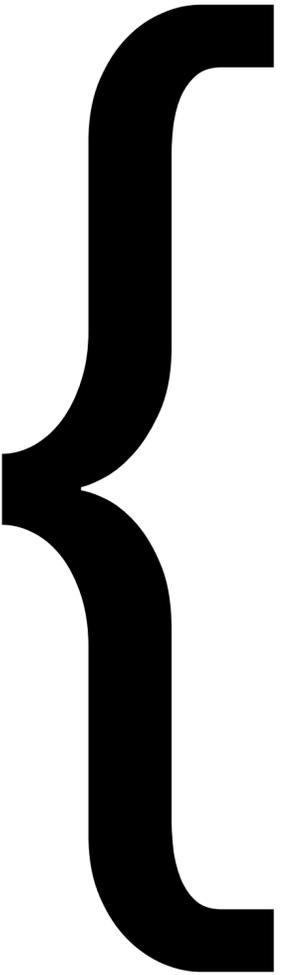
AV traps `0xc0000005`

vast amount of legacy code (really, really, really OLD code)

COM

Managed C++

non-standard C++



Challenges bringing ASan to Windows

the surface area of the Microsoft platform is enormous

non-standard C++

Structured Exception Handling (SEH) /EHa

AV traps 0xc0000005

vast amount of legacy code (really, really, really OLD code)

COM

Managed C++

ASan runtime interop with managed code (.NET)

Visual Studio ASan Experimental

**"Thank you" to Microsoft team*
tirelessly working on this**





2020: The Year of Sanitizers





Everyone will continue to invest heavily in this area (**sanitizers**) just because it's **so effective** at just finding correctness issues

Microsoft has contributed back to LLVM
all the work they've done to make ASan runtime work on Windows

github.com/llvm/llvm-project/tree/master/compiler-rt

Visual Studio 2019

ASan Visual Studio integration:

- **MSBuild & CMake** support for both Windows & Linux
- **Debugger** integration for MSVC and Clang/LLVM

aka.ms/asan

Address Sanitizer (ASan)

The screenshot shows a C++ IDE with a file named `ConsoleApplication6.cpp`. The code is as follows:

```
1  #include <iostream>
2
3  int main()
4  {
5      int* array = new int[100];
6      array[100] = 1;
7  }
```

Line 6, `array[100] = 1;`, is underlined with a red wavy line and has a red 'X' icon next to it. A tooltip window titled "Exception Unhandled" is open over this line, displaying the following text:

Exception Unhandled

Address Sanitizer Error: Heap buffer overflow

Full error details can be found in the output window

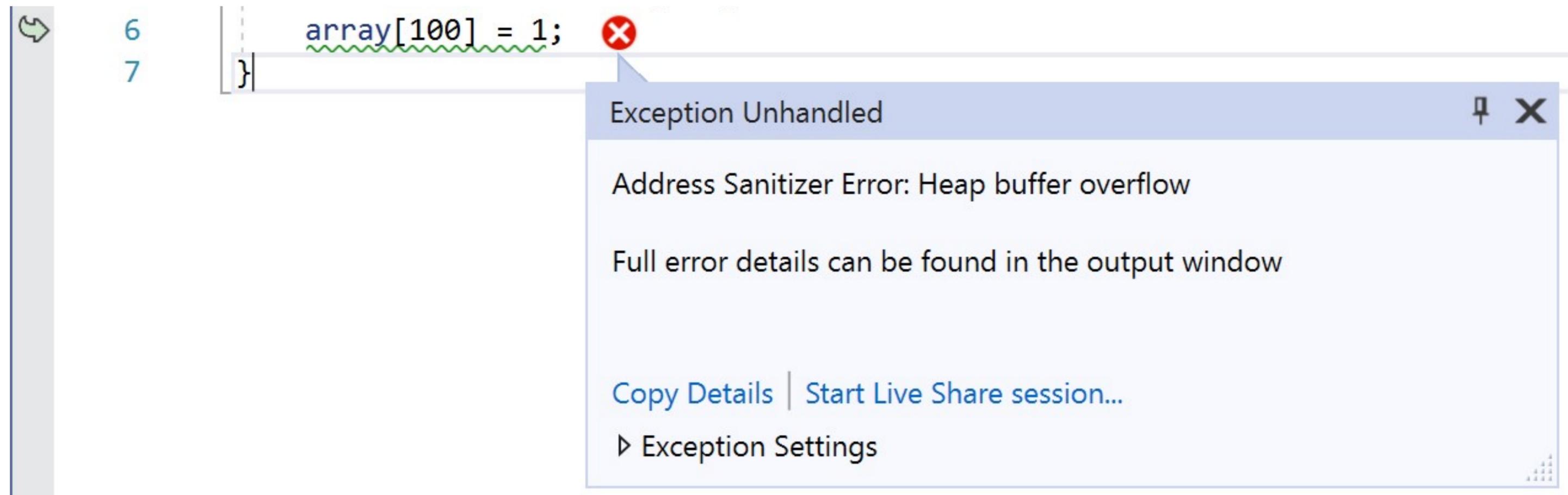
[Copy Details](#) | [Start Live Share session...](#)

▶ Exception Settings

Address Sanitizer (ASan)

IDE Exception Helper will be displayed when an issue is encountered
=> program execution will stop

ASan logging information => **Output window**



Clang/LLVM

```
==27748==ERROR: AddressSanitizer: stack-use-after-scope on address 0x0055fc68 at pc 0x793d62de bp 0x0055fbf4 sp 0x0055fbe8
WRITE of size 80 at 0x0055fc68 thread T0
#0 0x793d62f6 in __asan_wrap_memset d:\_work\5\s\llvm\projects\compiler-rt\lib\sanitizer_common\sanitizer_common_interceptors.inc:764
#1 0x77dd46e7 (C:\WINDOWS\SYSTEM32\ntdll.dll+0x4b2c46e7)
#2 0x77dd4ce1 (C:\WINDOWS\SYSTEM32\ntdll.dll+0x4b2c4ce1)
#3 0x75d408fe (C:\WINDOWS\System32\KERNELBASE.dll+0x100f08fe)
#4 0xa5ada0 in try_get_first_available_module minkernel\crts\ucrt\src\appcrt\internal\winapi_thunks.cpp:271
#5 0xa5ae99 in try_get_function minkernel\crts\ucrt\src\appcrt\internal\winapi_thunks.cpp:326
#6 0xa5b028 in __acrt_AppPolicyGetProcessTerminationMethodInternal minkernel\crts\ucrt\src\appcrt\internal\winapi_thunks.cpp:737
#7 0xa606ad in __acrt_get_process_end_policy minkernel\crts\ucrt\src\appcrt\internal\win_policies.cpp:84
#8 0xa52dcb in exit_or_terminate_process minkernel\crts\ucrt\src\appcrt\startup\exit.cpp:134
#9 0xa52da7 in common_exit minkernel\crts\ucrt\src\appcrt\startup\exit.cpp:280
#10 0xa52fb6 in exit minkernel\crts\ucrt\src\appcrt\startup\exit.cpp:293
#11 0xa2deb3 in _scrt_common_main_seh d:\agent\_work\2\s\src\vc\tools\crt\vcstartup\src\startup\exe_common.inl:295
#12 0x75ef6358 (C:\WINDOWS\System32\KERNEL32.DLL+0x6b816358)
#13 0x77df7a93 (C:\WINDOWS\SYSTEM32\ntdll.dll+0x4b2e7a93)
```

Address 0x0055fc68 is located in stack of thread T0
SUMMARY: AddressSanitizer: stack-use-after-scope d:\compiler-rt\lib\sanitizer_common\sanitizer_common_interceptors.inc:764 in __asan_wrap_memset

Shadow bytes around the buggy address:
0x300abf30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x300abf70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x300abf80: 00 00 00 00 00 00 00 00 00 00 00 00 00[f8]00 00
0x300abf90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x300abfd0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Shadow byte legend (one shadow byte represents 8 application bytes):

Addressable:	00
Partially addressable:	01 02 03 04 05 06 07
Heap left redzone:	fa
Freed heap region:	fd
Stack left redzone:	f1
Stack mid redzone:	f2
Stack right redzone:	f3
Stack after return:	f5
Stack use after scope:	f8
Global redzone:	f9
Global init order:	f6
Poisoned by user:	f7
Container overflow:	fc
Array cookie:	ac
Intra object redzone:	bb
ASan internal:	fe
Left alloca redzone:	ca
Right alloca redzone:	cb
Shadow gap:	cc

==27748==ABORTING

Snapshot File

Game changer!

Minidump file (*.dmp) <= Windows snapshot process (program virtual memory/heap + metadata)

VS can parse & open this => Points at the location the error occurred.

+ Live Share

Changes the way you report a bug, in general

The screenshot shows the 'Minidump File Summary' window in Visual Studio. It displays the following information:

- Minidump File Summary:** 11/5/2018 4:00:16 PM
- Dump Summary:**
 - Dump File: ShareSource.dmp : C:\User...
 - Last Write Time: 11/5/2018 4:00:16 PM
 - Process Name: ShareSource.exe : C:\Users\...
 - Process Architecture: x64
 - Exception Code: 0x80000004
 - Exception Information: A trace trap or other single-
 - Heap Information: Present
 - Error Information: [Expandable]
- System Information:**
 - OS Version: 10.0.17763
 - CLR Version(s): 4.6.26702.0
- Modules:**

Module Name	Module Version
ShareSource.exe	1.0.0.0
ntdll.dll	10.0.177
kernel32.dll	10.0.177

Actions available: Debug with Managed Only, Debug with Mixed, Debug with Native Only, Debug Managed Memory, Set symbol paths, Copy all to clipboard.



The screenshot shows the Visual Studio code editor with an exception dialog box open. The exception is 'ASAN Error: Stack Buffer Overflow' at line 124. The code snippet shows a loop with a 'double free' error. The 'Locals' window at the bottom shows the following variables:

Name	Value	Type
argc	2	int
argv	0x04301ad0 {0x04301adc "HeapCorruptionSample.e...}	char **
array	0x00cfff64 ""	char[256]
FileHandle	0x00000000	void *
freed_pointer	0x00000000	void *
readBytes	27	unsigned long

The 'Output' window shows memory dump data for the exception.

Visual Studio interface showing a C++ program with a stack buffer overflow exception. The code includes a loop that writes to an array, followed by a double-free attempt. An "Exception Unhandled" dialog box is open, displaying "ASAN Error: Stack Buffer Overflow" and a list of Azure Machine Learning buckets. A blue arrow points from the dialog to the output window, which shows memory addresses and hex values.

```

109   CloseHandle(FileHandle);
110
111   void* freed_pointer = malloc(1024);
112   free(freed_pointer); //we'll never get here either
113
114   if (array[0] == 'a') {
115       if (array[1] == 'b')
116           if (array[2] == 'c')
117               if (array[3] == 'd')
118                   if (array[4] == 'e')
119                       if (array[5] == 'f')
120                           if (array[6] == 'g')
121                               if (array[7] == 'h')
122                                   if (array[8] == 'i')
123                                       if (array[9] == 'j')
124                                           if (array[300] == 'X')
125                                               printf("we'll never get here either");
126
127   if (array[11] == 'k' && array[38] == 'g' && array[100] == 'b')
128   {
129       *((int*)freed_pointer) = 0x1c0debad; //uaf
130   }
131   else if (array[23] == '\xba')
132   {
133       free(freed_pointer); //double free
134   }
135
136   else if (strstr(array, "short"))
137   {
138       BYTE* byte_ptr = (BYTE*)malloc(1);
139   }

```

Exception Unhandled

ASAN Error: Stack Buffer Overflow

- [AzureMachine Bucket 0](#)
- [AzureMachine Bucket 1](#)
- [AzureMachine Bucket 2](#)
- [AzureMachine Bucket 3](#)
- [Manage Job Submission](#)

Full error details can be found in the output window
[Copy Details](#) | [Start collaboration session...](#)
 ▶ [Exception Settings](#)

Locals

Name	Value	Type
argc	2	int
argv	0x04301ad0 {0x04301adc "HeapCorruptionSample.e...}	char **
array	0x00cff6c4 ""	char[256]
FileHandle	0x00000000	void *
freed_pointer	0x00000000	void *
readBytes	27	unsigned long

Output

Show output from: Debug

```

0x3019fef0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x3019ff00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x3019ff10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 f1
0x3019ff20: f1 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x3019ff30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x3019ff40: 00 f2 f2 f2 f2 f2 04[f2]f8 f3 f3 f3 f3 00 00 00 00
0x3019ff50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x3019ff60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x3019ff70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x3019ff80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

```

Snapshot Loaded

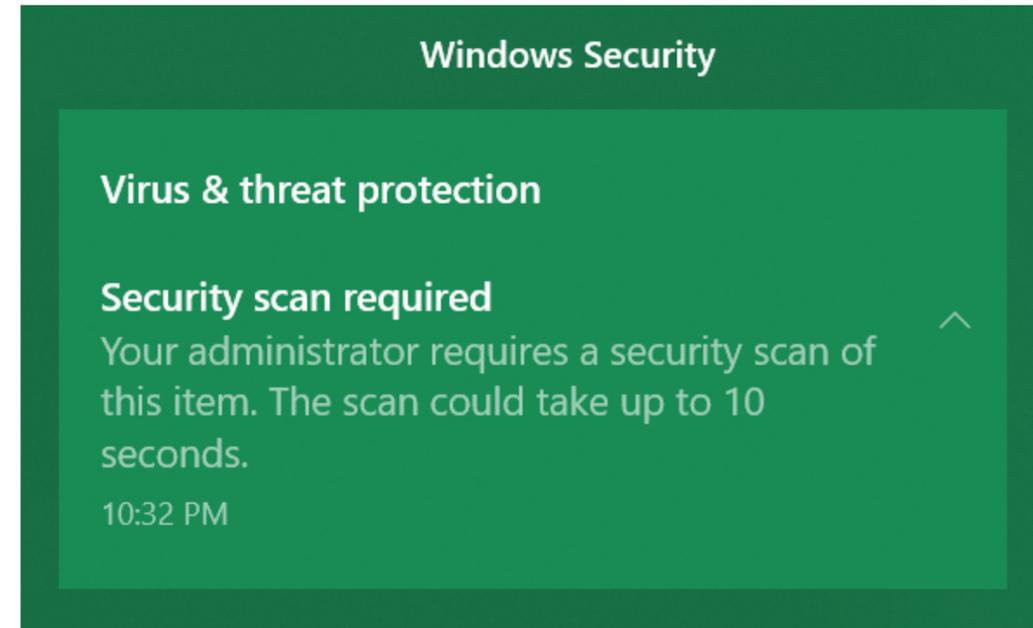
How does it work ?

ASan is just Malware, used for Good

```
Microsoft Visual Studio Debug Console
Hello World!
=====
==20932==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x12d3e28801d0 at pc 0x7ff6b4f21062 bp 0x00b85512f8b0
sp 0x00b85512f8b8
WRITE of size 4 at 0x12d3e28801d0 thread T0
==20932==WARNING: Failed to use and restart external symbolizer!
#0 0x7ff6b4f21061 in main C:\Users\victo\Downloads\Asana\Asana.cpp:10
#1 0x7ff6b4f22d03 in __scrt_common_main_seh D:\agent_work\9\s\src\vc\tools\crt\vcstartup\src\startup\exe_common.inl:
288
#2 0x7ffee9a76fd3 in BaseThreadInitThunk+0x13 (C:\WINDOWS\System32\KERNEL32.DLL+0x180016fd3)
#3 0x7ffeea97cec0 in RtlUserThreadStart+0x20 (C:\WINDOWS\SYSTEM32\ntdll.dll+0x18004cec0)
0x12d3e28801d0 is located 0 bytes to the right of 400-byte region [0x12d3e2880040,0x12d3e28801d0)
allocated by thread T0 here:
#0 0x7ffe889d7cf1 in _asan_loadN_noabort+0x553fb (C:\Program Files (x86)\Microsoft Visual Studio\2019\Professional\VC
Tools\MSVC\14.27.29110\bin\HostX64\x64\clang_rt.asan_dynamic-x86_64.dll+0x180057cf1)
#1 0x7ff6b4f21037 in main C:\Users\victo\Downloads\Asana\Asana.cpp:10
#2 0x7ff6b4f22d03 in __scrt_common_main_seh D:\agent_work\9\s\src\vc\tools\crt\vcstartup\src\startup\exe_common.inl:
288
#3 0x7ffee9a76fd3 in BaseThreadInitThunk+0x13 (C:\WINDOWS\System32\KERNEL32.DLL+0x180016fd3)
#4 0x7ffeea97cec0 in RtlUserThreadStart+0x20 (C:\WINDOWS\SYSTEM32\ntdll.dll+0x18004cec0)
SUMMARY: AddressSanitizer: heap-buffer-overflow C:\Users\victo\Downloads\Asana\Asana.cpp:10 in main
Shadow bytes around the buggy address:
0x05065ed88ffe0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x05065ed88fff0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x05065ed900000: fa fa fa fa fa fa fa fa 00 00 00 00 00 00 00 00
0x05065ed900010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x05065ed900020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x05065ed900030: 00 00 00 00 00 00 00 00 00[fa]fa fa fa fa fa
0x05065ed900040: fa fa
0x05065ed900050: fa fa
0x05065ed900060: fa fa
0x05065ed900070: fa fa
0x05065ed900080: fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
Addressable:          00
Partially addressable: 01 02 03 04 05 06 07
Heap left redzone:    fa
Freed heap region:    fd
Stack left redzone:   f1
Stack mid redzone:    f2
Stack right redzone:  f3
Stack after return:   f5
Stack use after scope: f8
Global redzone:       f9
Global init order:    f6
Poisoned by user:     f7
Container overflow:   fc
Array cookie:          ac
Intra object redzone: bb
ASan internal:         fe
Left alloca redzone:  ca
Right alloca redzone: cb
Shadow gap:           cc
==20932==ABORTING
C:\Users\victo\Downloads\Asana\x64\Release\Asana.exe (process 20932) exited with code 1.
Press any key to close this window . . .
```

ASan is just Malware, used for Good

```
Microsoft Visual Studio Debug Console
Hello World!
=====
==20932==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x12d3e28801d0 at pc 0x7ff6b4f21062 bp 0x00b85512f8b0
sp 0x00b85512f8b8
WRITE of size 4 at 0x12d3e28801d0 thread T0
==20932==WARNING: Failed to use and restart external symbolizer!
#0 0x7ff6b4f21061 in main C:\Users\victo\Downloads\Asana\Asana.cpp:10
#1 0x7ff6b4f22d03 in __scrt_common_main_seh D:\agent_work\9\s\src\vc\tools\crt\vcstartup\src\startup\exe_common.inl:
288
#2 0x7ffee9a76fd3 in BaseThreadInitThunk+0x13 (C:\WINDOWS\System32\KERNEL32.DLL+0x180016fd3)
#3 0x7ffeea97cec0 in RtlUserThreadStart+0x20 (C:\WINDOWS\SYSTEM32\ntdll.dll+0x18004cec0)
0x12d3e28801d0 is located 0 bytes to the right of 400-byte region [0x12d3e2880040,0x12d3e28801d0)
allocated by thread T0 here:
#0 0x7ffe889d7cf1 in _asan_loadN_noabort+0x553fb (C:\Program Files (x86)\Microsoft Visual Studio\2019\Professional\VC
\Tools\MSVC\14.27.29110\bin\HostX64\x64\clang_rt.asan_dynamic-x86_64.dll+0x180057cf1)
#1 0x7ff6b4f21037 in main C:\Users\victo\Downloads\Asana\Asana.cpp:10
#2 0x7ff6b4f22d03 in __scrt_common_main_seh D:\agent_work\9\s\src\vc\tools\crt\vcstartup\src\startup\exe_common.inl:
288
#3 0x7ffee9a76fd3 in BaseThreadInitThunk+0x13 (C:\WINDOWS\System32\KERNEL32.DLL+0x180016fd3)
#4 0x7ffeea97cec0 in RtlUserThreadStart+0x20 (C:\WINDOWS\SYSTEM32\ntdll.dll+0x18004cec0)
SUMMARY: AddressSanitizer: heap-buffer-overflow C:\Users\victo\Downloads\Asana\Asana.cpp:10 in main
Shadow bytes around the buggy address:
 0x05065ed8ffe0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 0x05065ed8fff0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 0x05065ed90000: fa fa fa fa fa fa fa fa 00 00 00 00 00 00 00 00
 0x05065ed90010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 0x05065ed90020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x05065ed90030: 00 00 00 00 00 00 00 00 00[fa]fa fa fa fa fa
 0x05065ed90040: fa fa
 0x05065ed90050: fa fa
 0x05065ed90060: fa fa
 0x05065ed90070: fa fa
 0x05065ed90080: fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
Addressable: 00
Partially addressable: 01 02 03 04 05 06 07
Heap left redzone: fa
Freed heap region: fd
Stack left redzone: f1
Stack mid redzone: f2
Stack right redzone: f3
Stack after return: f5
Stack use after scope: f8
Global redzone: f9
Global init order: f6
Poisoned by user: f7
Container overflow: fc
Array cookie: ac
Intra object redzone: bb
ASan internal: fe
Left alloca redzone: ca
Right alloca redzone: cb
Shadow gap: cc
==20932==ABORTING
C:\Users\victo\Downloads\Asana\x64\Release\Asana.exe (process 20932) exited with code 1.
Press any key to close this window . . .
```



Address Sanitizer (ASan)

Compiler

- instrumentation code, stack layout, and calls into runtime
- meta-data in OBJ for the runtime

Sanitizer Runtime

- hooking `malloc()`, `free()`, `memset()`, etc.
- error analysis and reporting
- does not require complete recompile => great for **interop**
- **zero** false positives

ASan Report

==23364==ERROR: AddressSanitizer: **heap-buffer-overflow** on address 0x12ac01b801d0 at
pc 0x7ff6e3a627be bp 0x0097d4b4fac0 sp 0x0097d4b4fac8

WRITE of size 4 at 0x12ac01b801d0 thread T0

```
#0 0x7ff6e3a627bd in main C:\Asana\Asana.cpp:10
#1 0x7ff6e3a66ce8 in invoke_main D:\agent\_work\9\s\src\vctools\crt\vcstartup\src\startup\exe_common.inl:78
#2 0x7ff6e3a66bcd in __scrt_common_main_seh D:\agent\_work\9\s\src\vctools\crt\vcstartup\src\startup\exe_common.inl:288
#3 0x7ff6e3a66a8d in __scrt_common_main D:\agent\_work\9\s\src\vctools\crt\vcstartup\src\startup\exe_common.inl:330
#4 0x7ff6e3a66d78 in mainCRTStartup D:\agent\_work\9\s\src\vctools\crt\vcstartup\src\startup\exe_main.cpp:16
#5 0x7ffee9a76fd3 in BaseThreadInitThunk+0x13 (C:\WINDOWS\System32\KERNEL32.DLL+0x180016fd3)
#6 0x7ffeea97cec0 in RtlUserThreadStart+0x20 (C:\WINDOWS\SYSTEM32\ntdll.dll+0x18004cec0)
```

0x12ac01b801d0 is located 0 bytes to the right of 400-byte region [0x12ac01b80040,0x12ac01b801d0)

allocated by thread T0 here:

```
#0 0x7ffe83be7e91 in _asan_loadN_noabort+0x55555 (...\.bin\HostX64\x64\clang_rt.asan_dbg_dynamic-x86_64.dll+0x180057e91)
#1 0x7ff6e3a62758 in main C:\Asana\Asana.cpp:9
#2 0x7ff6e3a66ce8 in invoke_main D:\agent\_work\9\s\src\vctools\crt\vcstartup\src\startup\exe_common.inl:78
#3 0x7ff6e3a66bcd in __scrt_common_main_seh D:\agent\_work\9\s\src\vctools\crt\vcstartup\src\startup\exe_common.inl:288
#4 0x7ff6e3a66a8d in __scrt_common_main D:\agent\_work\9\s\src\vctools\crt\vcstartup\src\startup\exe_common.inl:330
#5 0x7ff6e3a66d78 in mainCRTStartup D:\agent\_work\9\s\src\vctools\crt\vcstartup\src\startup\exe_main.cpp:16
#6 0x7ffee9a76fd3 in BaseThreadInitThunk+0x13 (C:\WINDOWS\System32\KERNEL32.DLL+0x180016fd3)
#7 0x7ffeea97cec0 in RtlUserThreadStart+0x20 (C:\WINDOWS\SYSTEM32\ntdll.dll+0x18004cec0)
```

SUMMARY: AddressSanitizer: [heap-buffer-overflow](#) C:\Asana\Asana.cpp:10 in main()

Shadow bytes around the buggy address:

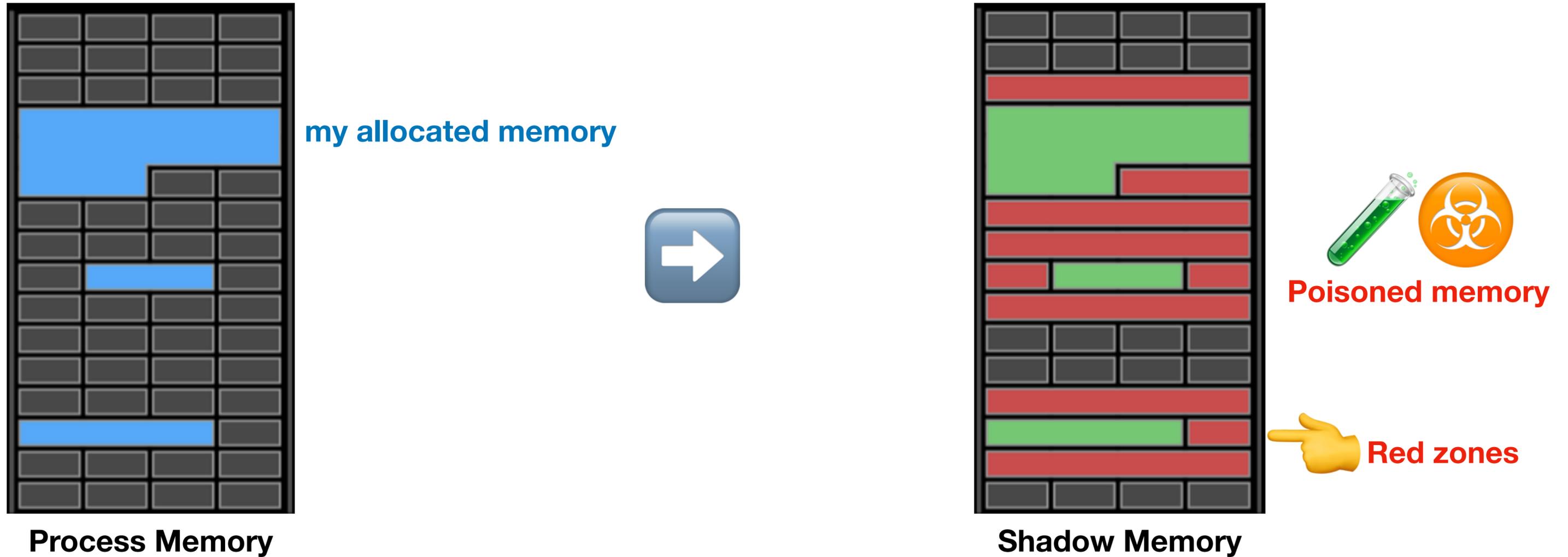
```
0x04d981eef0e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x04d981eef0f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x04d981ef0000: fa fa fa fa fa fa fa fa 00 00 00 00 00 00 00 00
0x04d981ef0010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x04d981ef0020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x04d981ef0030: 00 00 00 00 00 00 00 00 00 00 [fa] fa fa fa fa fa
0x04d981ef0040: fa fa
0x04d981ef0050: fa fa
0x04d981ef0060: fa fa
0x04d981ef0070: fa fa
0x04d981ef0080: fa fa
```

Addressable:	00	👍	
Partially addressable:	01	02	03 04 05 06 07 (of the 8 application bytes, how many are accessible)
Heap left redzone:	fa	←	
Freed heap region:	fd		
Stack left redzone:	f1		
Stack mid redzone:	f2		
Stack right redzone:	f3		
Stack after return:	f5		
Stack use after scope:	f8		
Global redzone:	f9		issues & markers
Global init order:	f6		
Poisoned by user:	f7		
Container overflow:	fc		
Array cookie:	ac		
Intra object redzone:	bb		
ASan internal:	fe		
Left alloca redzone:	ca		
Right alloca redzone:	cb		
Shadow gap:	cc	←	

Shadow byte legend

(one shadow byte represents 8 application bytes)

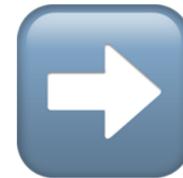
Shadow Mapping



Code Generation

(simplified)

```
*p = 0xbadf00d
```



```
if (ShadowByte::IsBad(p))  
    AsanRt::Report(p, sz)
```

```
*p = 0xbadf00d
```

If the shadow byte is **poisoned**,
ASAN runtime **reports** the problem and **crashes** the application

Code Generation

(simplified)

Lookups into shadow memory need to be *very fast*

ASAN maintains a *lookup table* where every **8 bytes** of user memory are tracked by **1 shadow byte**

=> **1/8** of the address space (*shadow region*)

A Shadow Byte: $*((User_Address \gg 3) + 0x30000000) = 0xF8;$

↑
Stack use after scope

Code Generation (simplified)

Lookups into shadow memory need to be **very fast**

```
bool ShadowByte::IsBad(Addr) // is poisoned ?  
{  
    Shadow = Addr >> 3 + Offset;  
    return (*Shadow) != 0;  
}
```

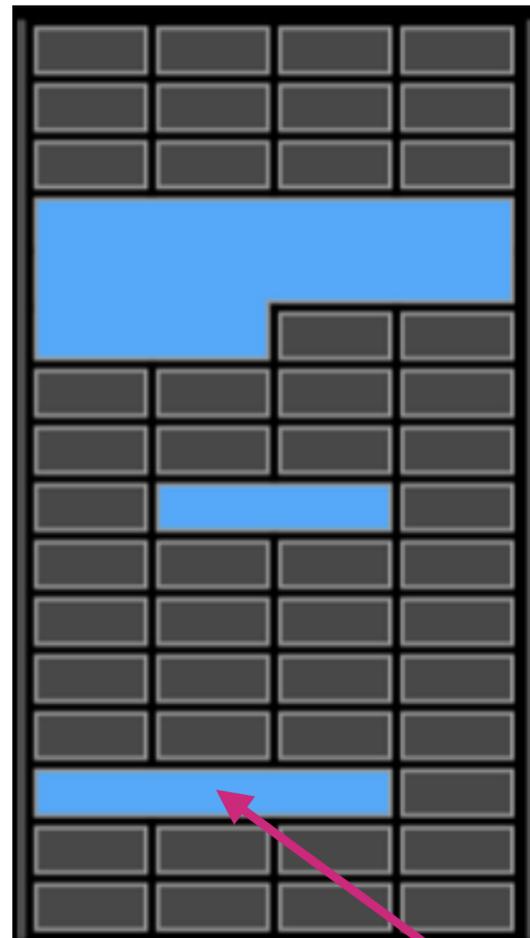
Location of shadow region in memory

A Shadow Byte:

```
*( (User_Address >> 3) + 0x30000000 ) = 0xF8;
```

Stack use after scope

Shadow Mapping

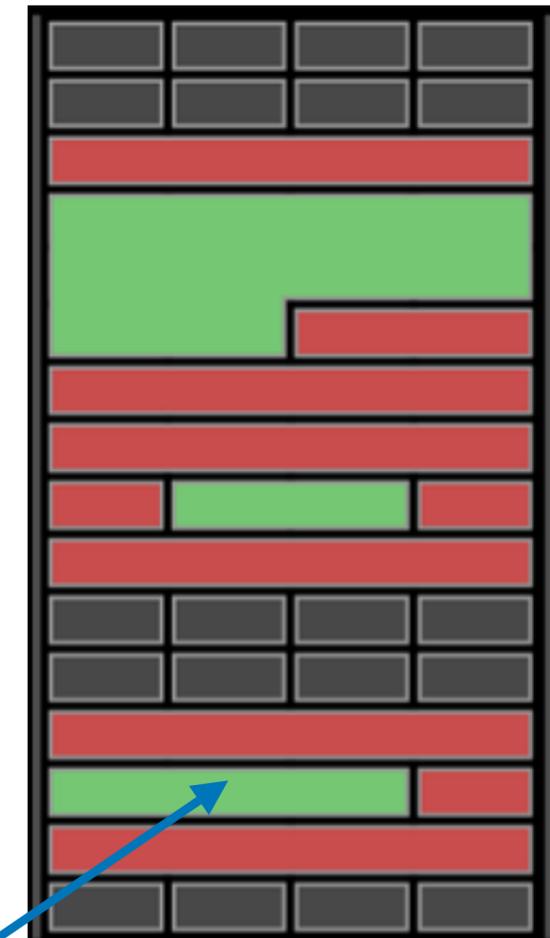


Process Memory

p

```
if (ShadowByte::IsBad(p))  
    AsanRt::Report(p, sz);
```

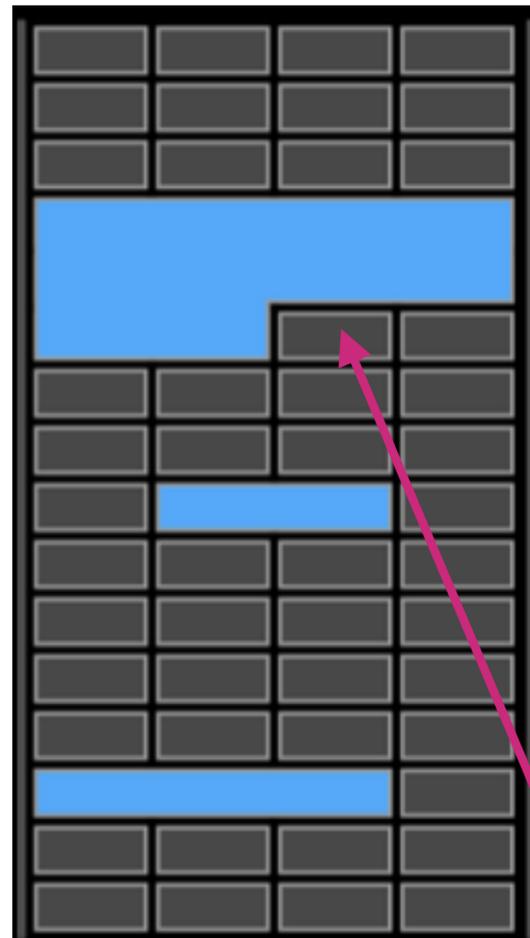
*p = 0xf00d



Shadow Memory

ShadowByte(p)

Shadow Mapping

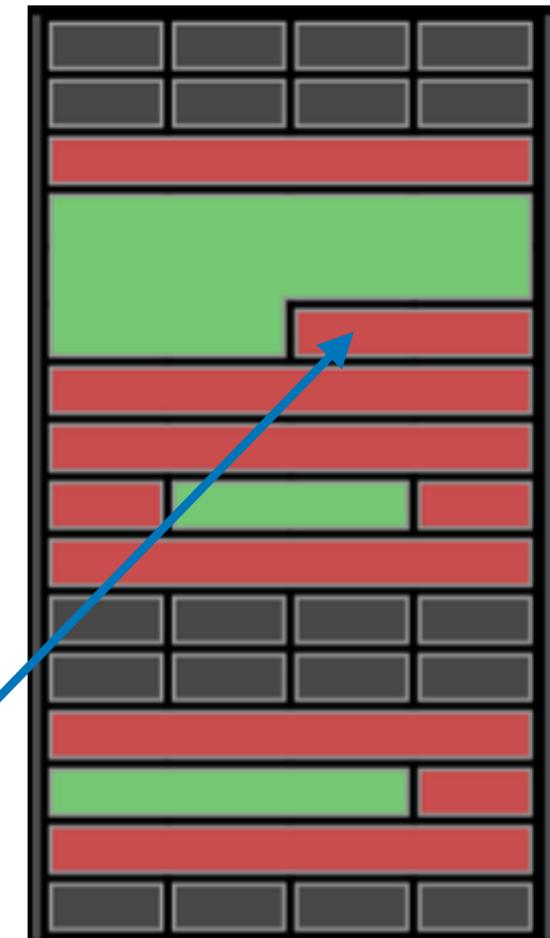


Process Memory

p

```
if (ShadowByte::IsBad(p))  
    AsanRt::Report(p, sz);
```

```
*p = 0xbadf00d
```



Shadow Memory

ShadowByte(p)

Heap Red Zones

malloc()



ASAN malloc()

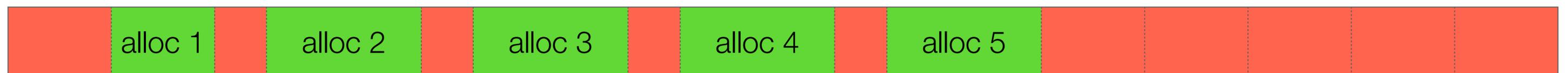


Heap Red Zones

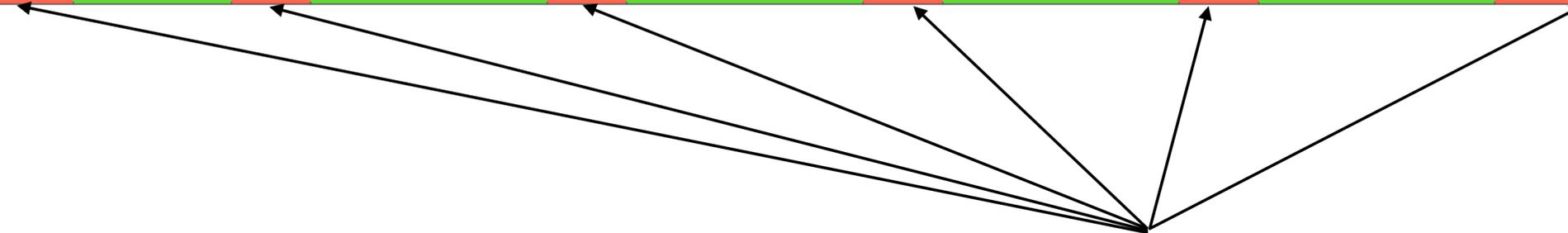
ASAN malloc()



Shadow Memory



Poisoned memory



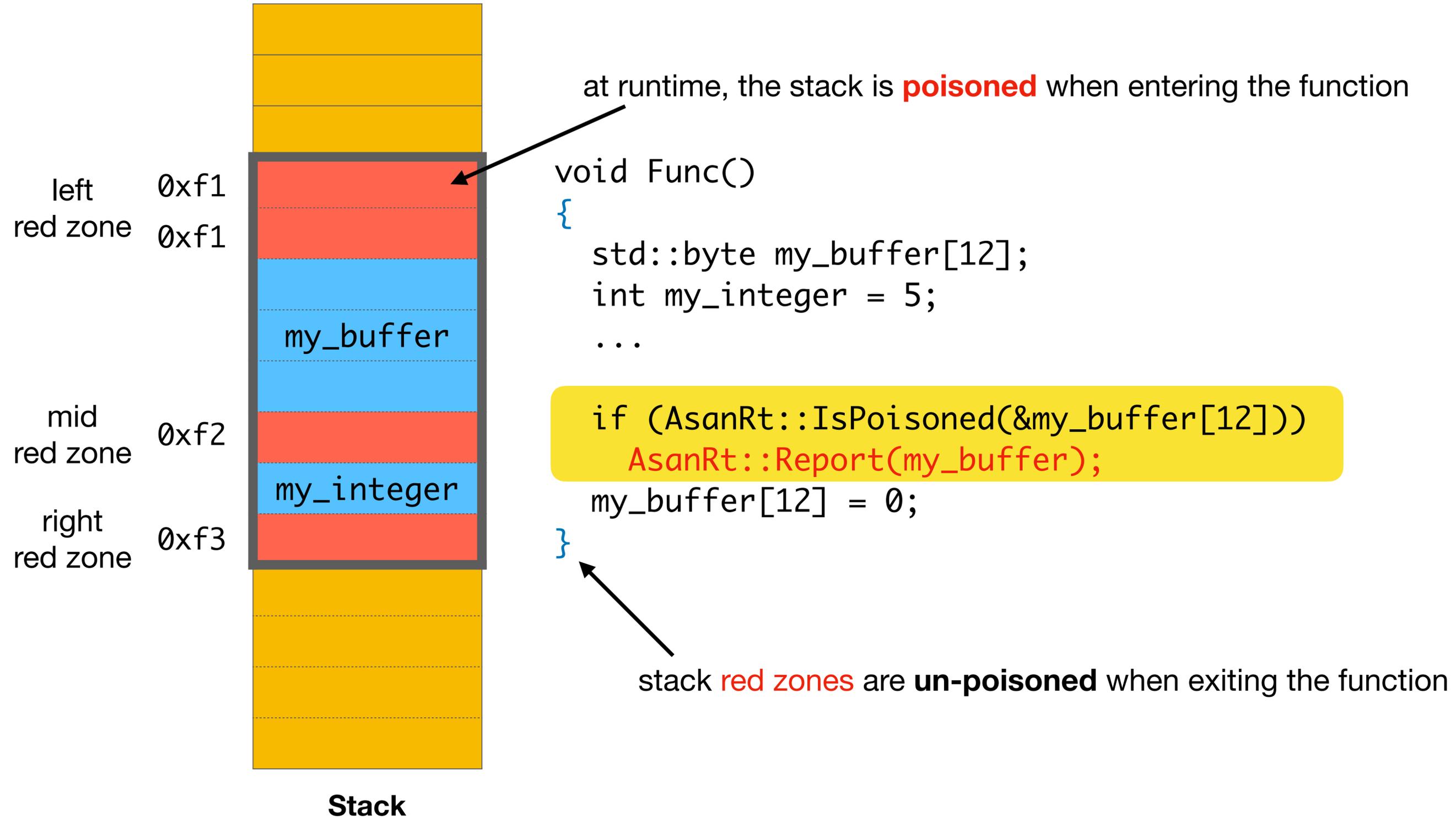
Stack Red Zones



Stack

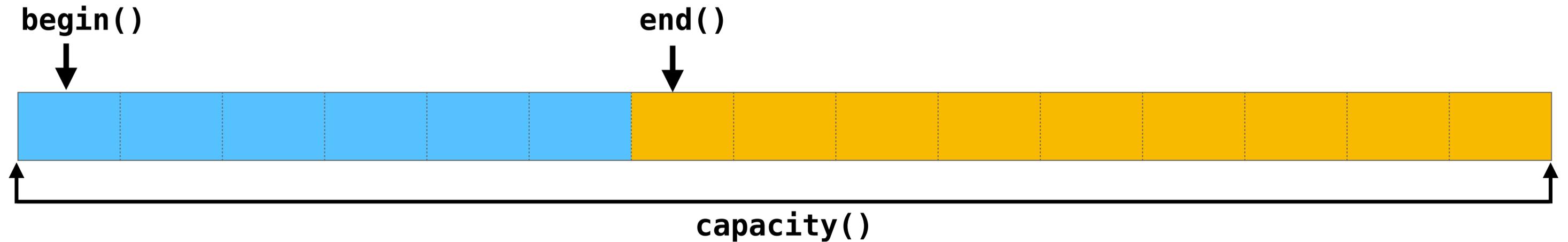
```
void Func()
{
    std::byte my_buffer[12];
    int my_integer = 5;
    ...
    ...
    ...
    ...
    my_buffer[12] = 0;
}
```

Stack Red Zones



AddressSanitizer ContainerOverflow

`std::vector<T>`

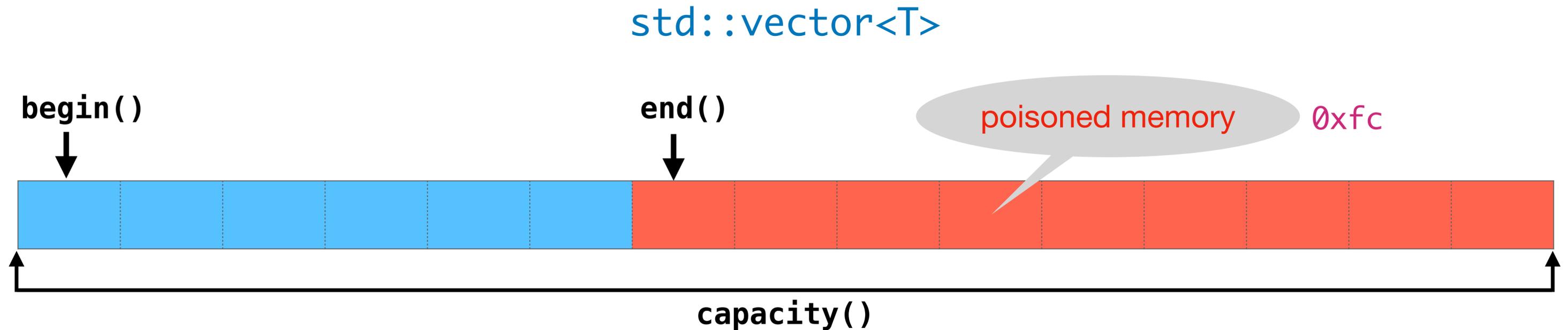


with the help of **code annotations** in `std::vector`

libc++
libstdc++

<https://github.com/google/sanitizers/wiki/AddressSanitizerContainerOverflow>

AddressSanitizer ContainerOverflow



```
std::vector<int> v;  
v.push_back(0);  
v.push_back(1);  
v.push_back(2);  
assert(v.capacity() >= 4);  
assert(v.size() == 3);
```

```
T * p = &v[0];  
std::cout << p[3];
```

container-overflow

0xfc

v[3] could be detected by
simple checks in std::vector

<https://github.com/google/sanitizers/wiki/AddressSanitizerContainerOverflow>



Address Sanitizer (ASan)

Very fast instrumentation

The average slowdown of the instrumented program is $\sim 2x$

github.com/google/sanitizers/wiki/AddressSanitizerPerformanceNumbers

Problems & Gotchas

Stuff you need to know

VS 16.7.x-16.8.Preview

Compiling/linking from command-line

Manual CLI compile/link can be tedious,
be careful in choosing the correct **ASan libraries** to link against

Check here for all the details:

devblogs.microsoft.com/cppblog/asan-for-windows-x64-and-debug-build-support/

Eg.

- **Compiling a single static EXE**
link the static runtime `asan-i386.lib` and the cxx library
- **Compiling an EXE with /MT runtime which will use ASan-instrumented DLLs**
the EXE needs to have `asan-i386.lib` linked and
the DLLs need the `clang_rt.asan_dll_thunk-i386.lib`
- **When compiling with the /MD dynamic runtime**
all EXE and DLLs with instrumentation should be linked with
`asan_dynamic-i386.lib` and `clang_rt.asan_dynamic_runtime_thunk-i386.lib`
At runtime, these libraries will refer to the
`clang_rt.asan_dynamic-i386.dll` shared ASan runtime.

`/ZI`

Edit and Continue (Debug)

error MSB8059:

`-fsanitize=address` (Enable Address Sanitizer) is incompatible with option `'edit-and-continue'` debug information `/ZI`

Mixing ASan & non-ASan modules

Problem:

A non-ASan built executable can NOT call `LoadLibrary()` on a DLL built with ASAN.

Reason:

ASan runtime is tracking memory and the non-ASan executable might have done something like `HeapAlloc()`

This limitation is a problem if you're building a plugin (DLL)

MSVC team is considering dealing with this issue in a later release

devblogs.microsoft.com/cppblog/asan-for-windows-x64-and-debug-build-support/

/RTCs and /RTC1 Runtime Checks

warning C5059:

runtime checks and address sanitizer is not currently supported - disabling runtime checks

If you use `/WX` this harmless/informative warning becomes a build blocker :(

=> we had to disable `/RTCs` and `/RTC1` so we could do the ASan experiments



twitter.com/ciura_victor/status/1296499633825492992

Missing PDBs from VS

It appears some ASan runtime PDBs were not included in the VS installer:

```
[Debug]  
vcasand.lib(vcasan.obj) : warning LNK4099: PDB 'vcasand.pdb' was not found with 'vcasand.lib(vcasan.obj)'  
linking object as if no debug info
```

```
[Release]  
vcasan.lib(vcasan.obj) : warning LNK4099: PDB 'vcasan.pdb' was not found with 'vcasan.lib(vcasan.obj)'  
linking object as if no debug info
```

Building an EXE

Missing PDBs from VS

It appears some PDBs were not included in the VS installer:

[Debug]

```
libvcasand.lib(vcasan.obj) : warning LNK4099: PDB 'libvcasand.pdb' was not found with  
'libvcasand.lib(vcasan.obj)'
```

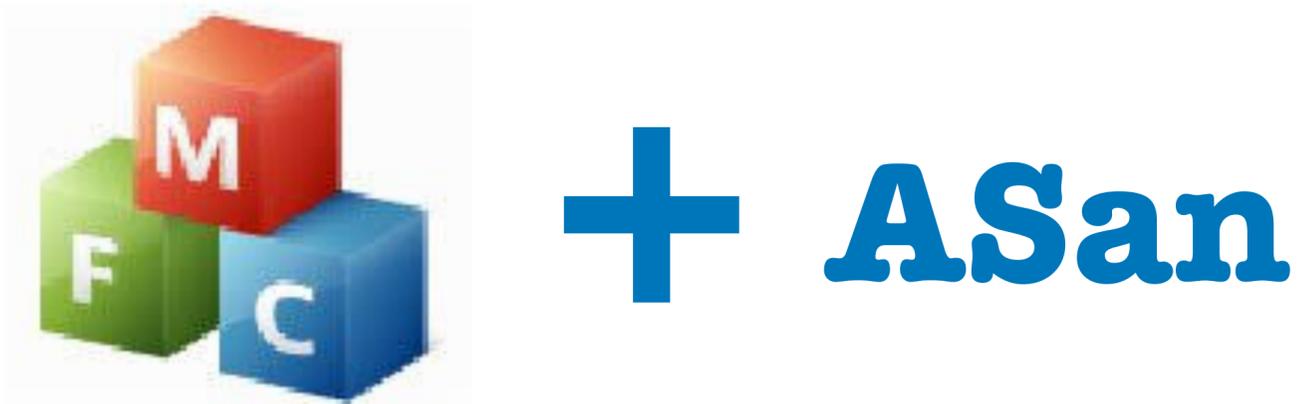
[Release]

```
libvcasan.lib(vcasan.obj) : warning LNK4099: PDB 'libvcasan.pdb' was not found with  
'libvcasan.lib(vcasan.obj)'
```

Building a static LIB, linked into an EXE

vcasan(d).lib

- creates **metadata** the **IDE** will parse to support error reporting in its sub-panes
- metadata is stored in **.dmp** files produced when a program is terminated by ASan



```
>uafxcw.lib(afxmem.obj) : error LNK2005: "void * __cdecl operator new(unsigned int)" (??2@YAPAXI@Z) already defined in clang_rt.asan_cxx-i386.lib(asan_new_delete.cc.obj)
```

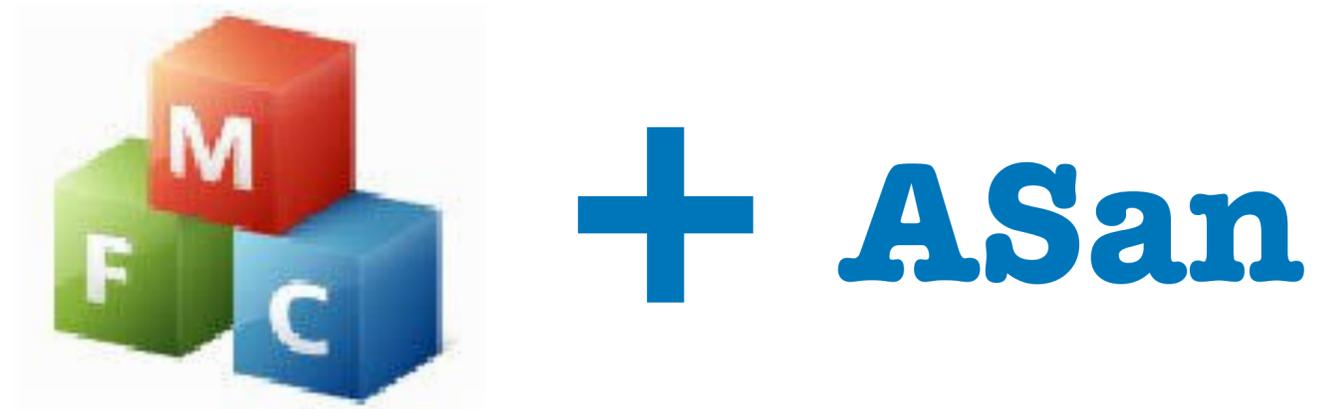
```
>uafxcw.lib(afxmem.obj) : error LNK2005: "void __cdecl operator delete(void *)" (??3@YAXPAX@Z) already defined in clang_rt.asan_cxx-i386.lib(asan_new_delete.cc.obj)
```

```
>uafxcw.lib(afxmem.obj) : error LNK2005: "void * __cdecl operator new[](unsigned int)" (??_U@YAPAXI@Z) already defined in clang_rt.asan_cxx-i386.lib(asan_new_delete.cc.obj)
```

```
>uafxcw.lib(afxmem.obj) : error LNK2005: "void __cdecl operator delete[](void *)" (??_V@YAXPAX@Z) already defined in clang_rt.asan_cxx-i386.lib(asan_new_delete.cc.obj)
```

 **if you link **statically** to MFC lib**

developercommunity.visualstudio.com/content/problem/1144525/mfc-application-fails-to-link-with-address-sanitiz.html



In general, if you have **overrides** for:

```
void* operator new(size_t size);
```

Workarounds:

- set `/FORCE:MULTIPLE` in the linker command line (settings)
- temporarily set your MFC application to link to **shared** MFC DLLs for testing with ASan



Explore Further

AddressSanitizer (ASan) for Windows with MSVC

devblogs.microsoft.com/cppblog/addresssanitizer-asan-for-windows-with-msvc/

AddressSanitizer for Windows: x64 and Debug Build Support

devblogs.microsoft.com/cppblog/asan-for-windows-x64-and-debug-build-support/

by **Augustin Popa**

[@augustin_popa](https://twitter.com/augustin_popa)

Part III

Warm Fuzzy Feelings

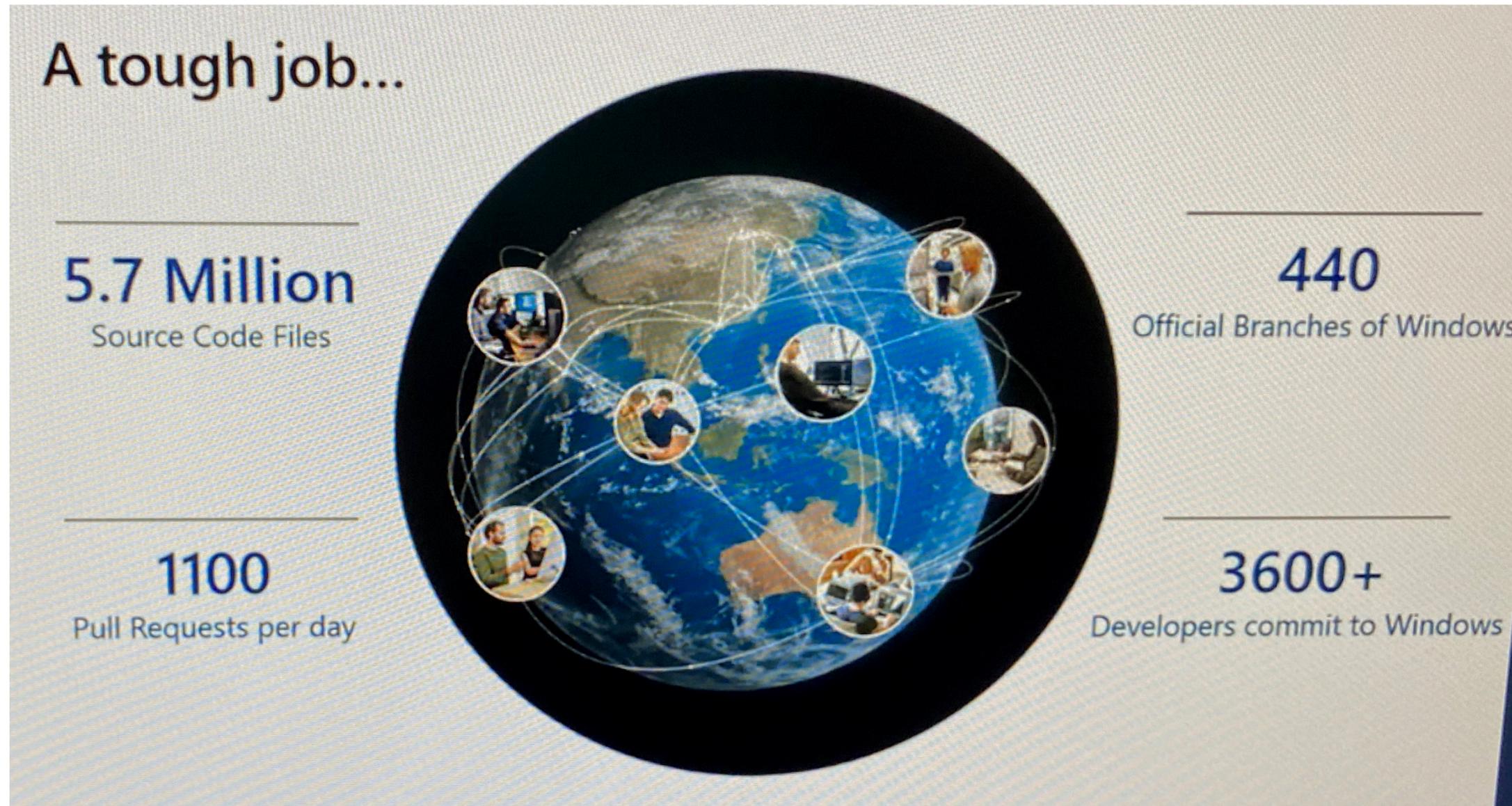
Sanitizers + Fuzzing



Automatically generate inputs to you program to crash it.

Sanitizers + Fuzzing

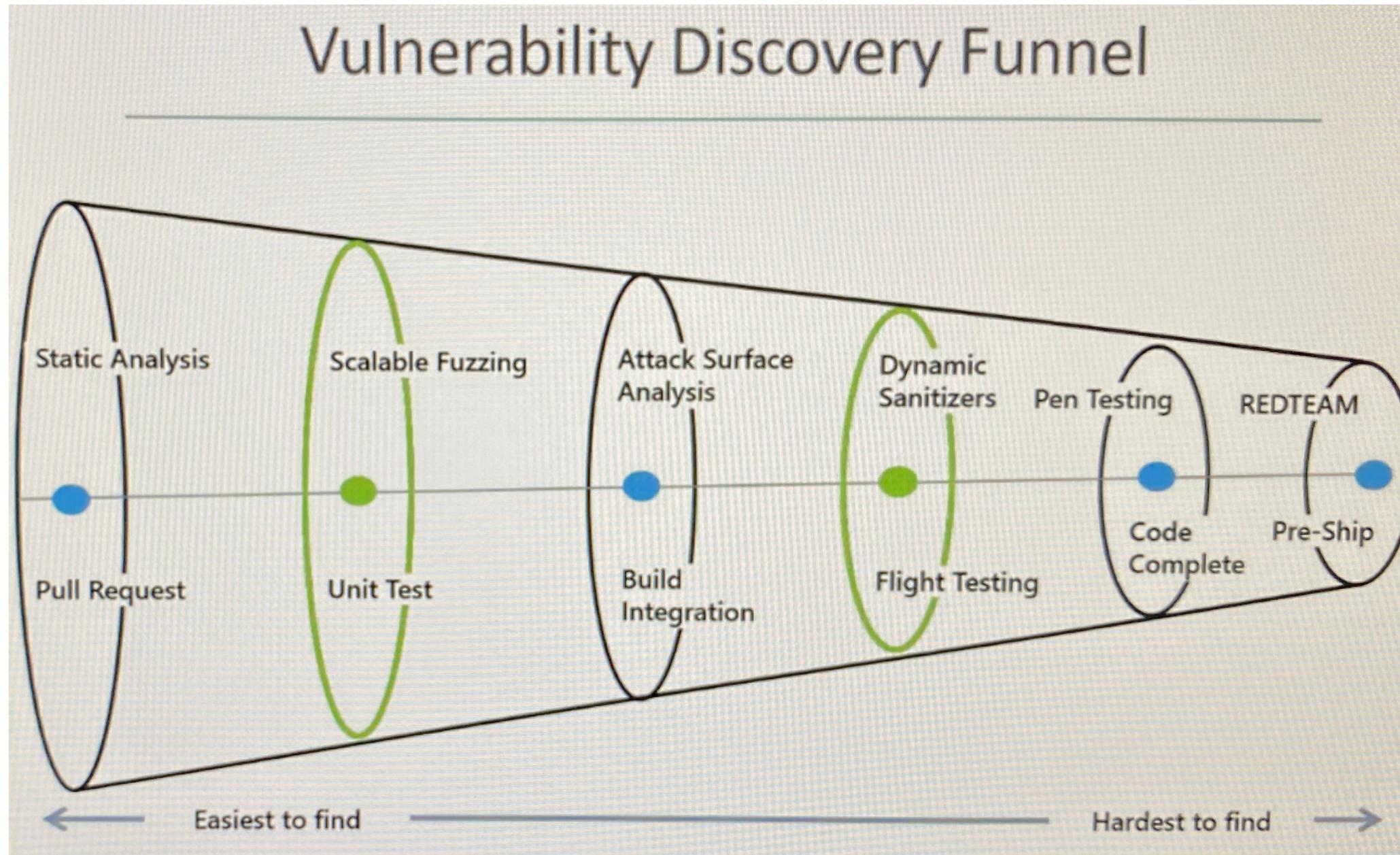
Case study at Microsoft Windows scale



<https://sched.co/e7C0>

Sanitizers + Fuzzing

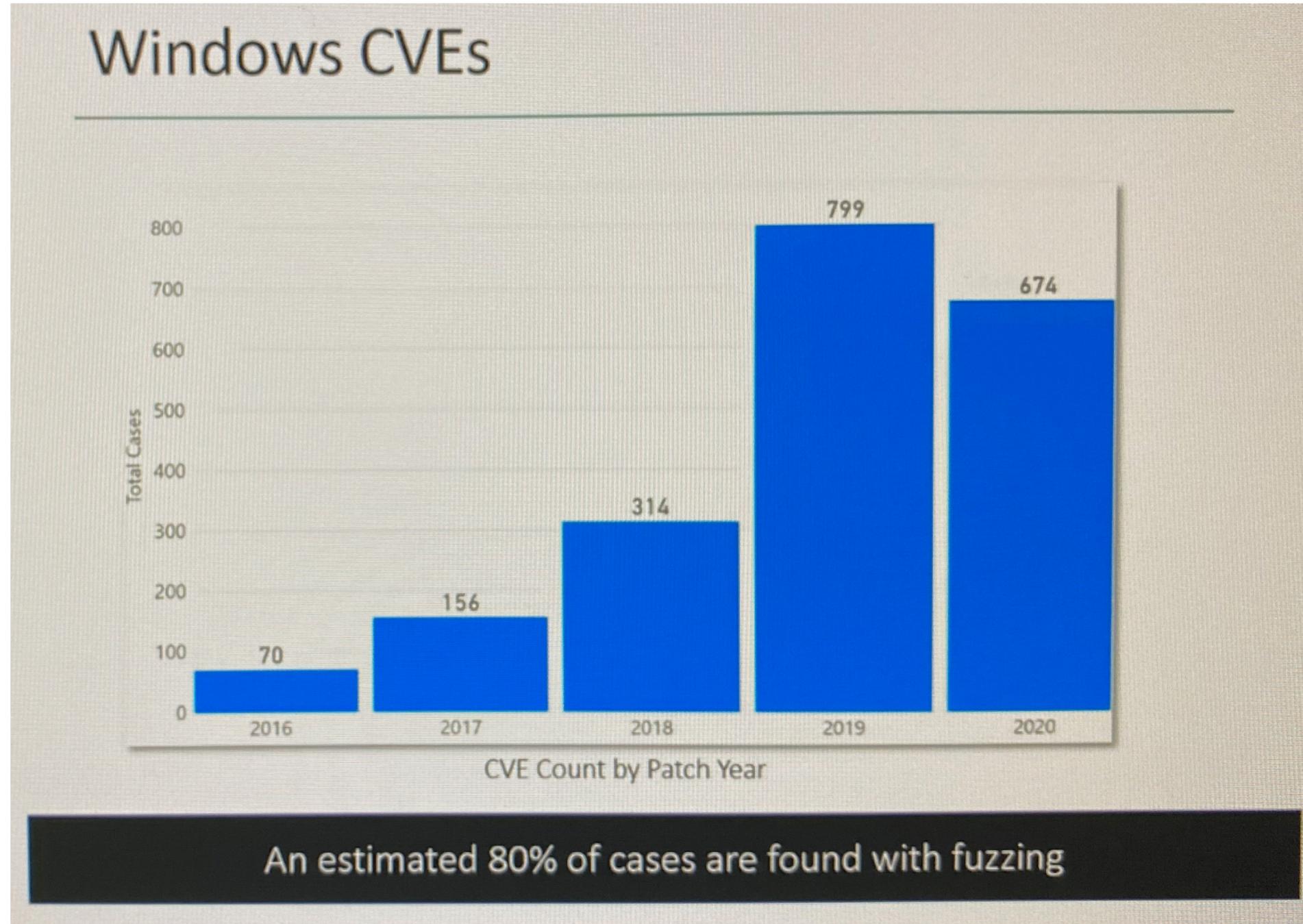
Case study at Microsoft Windows scale



<https://sched.co/e7C0>

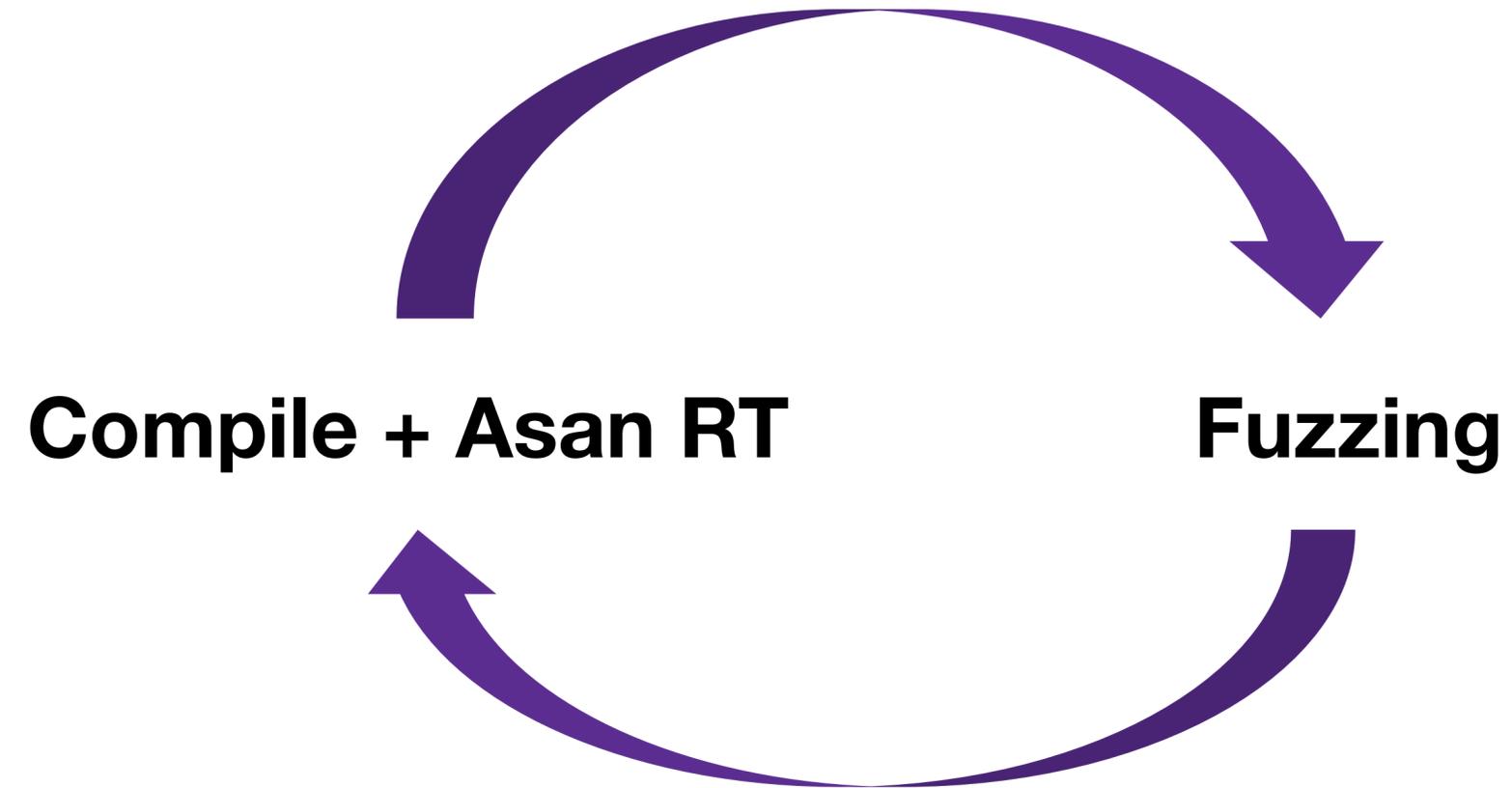
Sanitizers + Fuzzing

Case study at Microsoft Windows scale



<https://sched.co/e7C0>

Workflow





{ ASan + Fuzzing } => Azure

What is Microsoft Security Risk Detection?

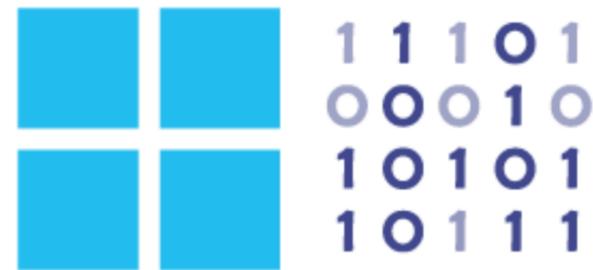
Security Risk Detection is Microsoft's unique fuzz testing service for finding security critical bugs in software. Security Risk Detection helps customers quickly adopt practices and technology battle-tested over the last 15 years at Microsoft.

[READ SUCCESS STORIES >](#)



"Million dollar" bugs

Security Risk Detection uses "Whitebox Fuzzing" technology which discovered 1/3rd of the "million dollar" security bugs during Windows 7 development.



Battle tested tech

The same state-of-the-art tools and practices honed at Microsoft for the last decade and instrumental in hardening Windows and Office — with the results to prove it.



Scalable fuzz lab in the cloud

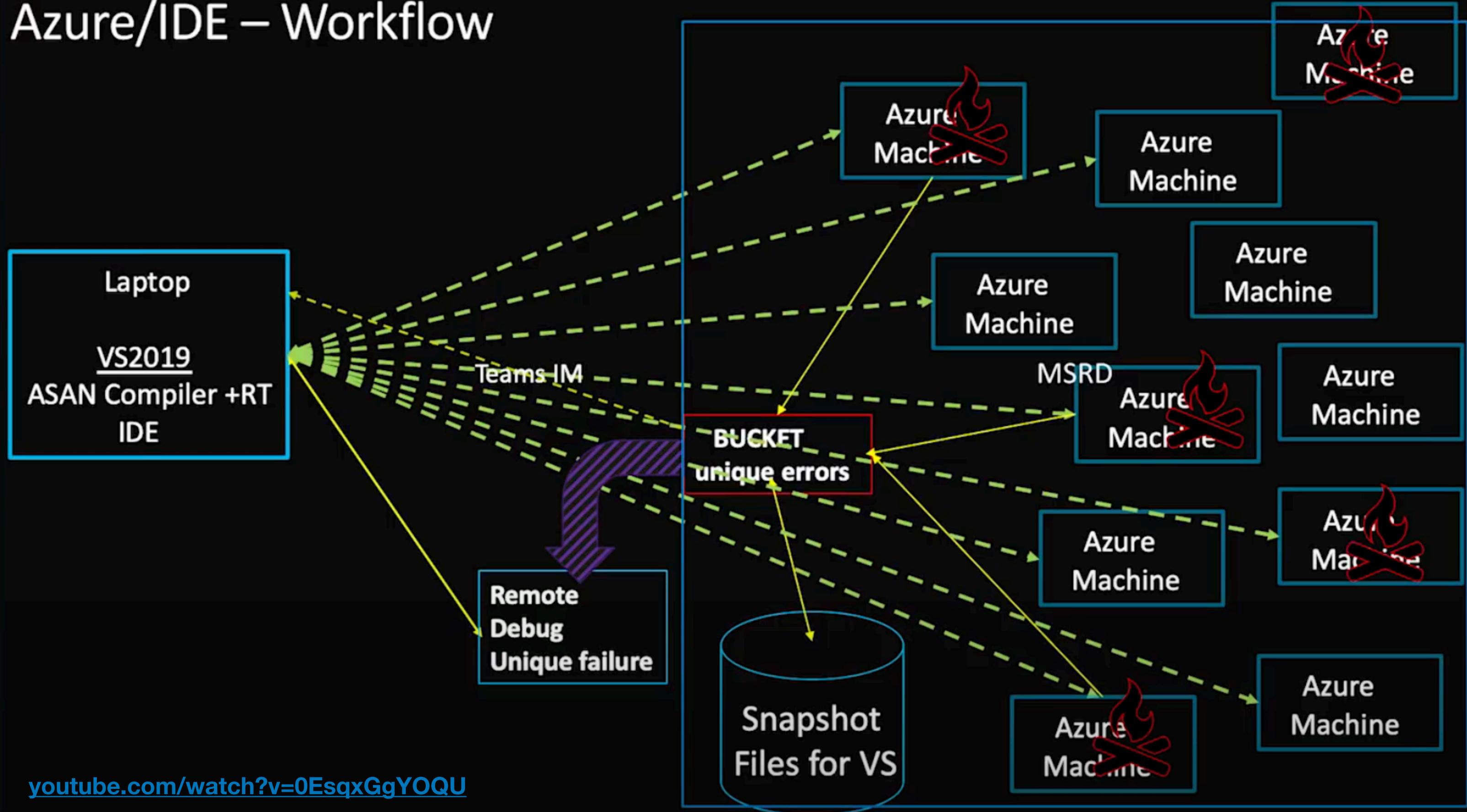
One click scalable, automated, Intelligent Security testing lab in the cloud.



Cross-platform support

Linux Fuzzing is now available. So, whether you're building or deploying software for Windows or Linux or both, you can utilize our Service.

Azure/IDE – Workflow



youtube.com/watch?v=0EsqxGgYOQU

Microsoft OneFuzz

a platform you will be able to download from **Github**
and run fuzzing on premise or in **Azure**



Introducing Project OneFuzz From Microsoft

```
... object to mirror
mirror_mod.mirror_object =
operation == "MIRROR_X":
mirror_mod.use_x = True
mirror_mod.use_y = False
mirror_mod.use_z = False
operation == "MIRROR_Y":
mirror_mod.use_x = False
mirror_mod.use_y = True
mirror_mod.use_z = False
operation == "MIRROR_Z":
mirror_mod.use_x = False
mirror_mod.use_y = False
mirror_mod.use_z = True

... selection at the end -add
..._ob.select= 1
..._ob.select=1
...context.scene.objects.active
("Selected" + str(modifier
...mirror_ob.select = 0
... bpy.context.selected_ob
...data.objects[one.name].se

...int("please select exactly

... OPERATOR CLASSES ...

...types.Operator):
... X mirror to the selected
...ect.mirror_mirror_x"
...ror X"
```

The code that fuzzes Windows continuously released today as MIT-Licensed Open Source for integration with your builds

Justin Campbell, Windows Security
Mike Walker, Microsoft Research

1 week ago

Project OneFuzz

September 15, 2020

Microsoft announces new Project OneFuzz framework, an open source developer tool to find and fix bugs at scale

Justin Campbell Principal Security Software Engineering Lead, Microsoft Security

Mike Walker Senior Director, Special Projects Management, Microsoft Security

A self-hosted Fuzzing-As-A-Service platform

microsoft.com/security/blog/2020/09/15/microsoft-onefuzz-framework-open-source-developer-tool-fix-bugs/

A self-hosted Fuzzing-As-A-Service platform

github.com/microsoft/onefuzz

Project OneFuzz

CI/CD



New unique crashes create notifications:

- **Teams**
- **ADO work items**



Azure DevOps Pipeline



GitHub Actions

github.com/microsoft/onefuzz-samples

{ ASan + Fuzzing } => Azure

Microsoft Security Risk Detection

Fuzzing Jobs

Fuzzing Jobs

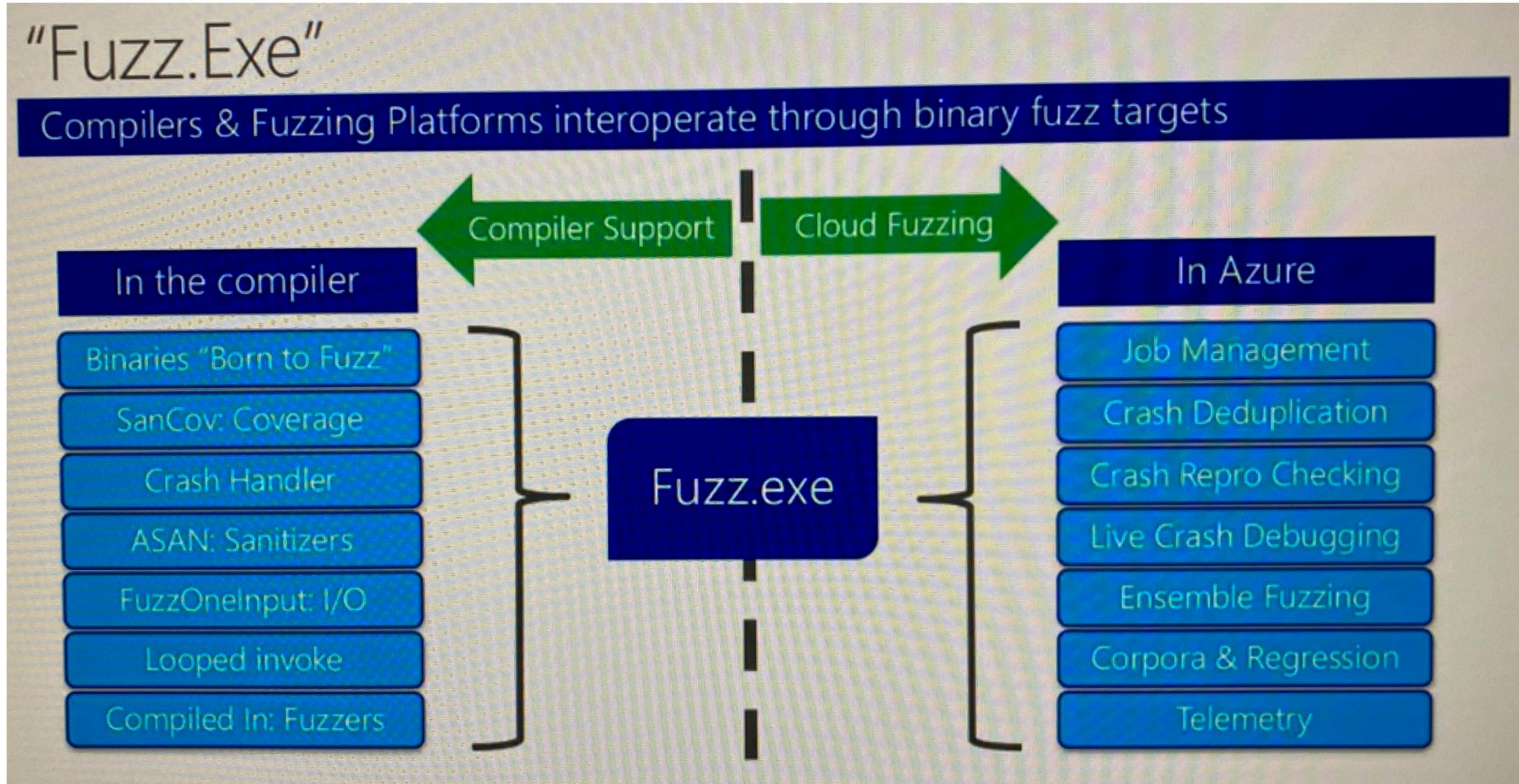
Create Job

Id	Name	OS Image	Created	Status	Results	Actions
8ee12290	Package CppConFuzzTargetVcAsan by jradigan from JRADIGAN-DELLLT	Windows Server 2019 Datacenter x64	9/18/19 1:44 PM	Fuzzing (Day 1 of 14) Started on: 9/18/19 2:09 PM	4	[Icons]
fb907d35	Package CppConFuzzTargetVcAsan by jradigan from JRADIGAN-DELLLT	Windows Server 2019 Datacenter x64	9/18/19 9:47 AM	Fuzzing (Day 1 of 14) Started on: 9/18/19 10:13 AM	5	[Icons]
b4058add	Package CppConFuzzTargetVcAsan by jradigan from JRADIGAN-DELLLT	Windows Server 2019 Datacenter x64	9/13/19 1:55 PM	Fuzzing (Day 5 of 14) Started on: 9/13/19 2:21 PM	5	[Icons]
6852ebcc	Package CppConFuzzTargetVcAsan	Windows Server 2019 Datacenter x64	9/13/19 9:11 AM	Stopped	5	[Icons]
9f1428c0	Demo - Package CppConFuzzTargetVcAsan	Windows Server 2019 Datacenter x64	9/8/19 7:27 AM	Fuzzing (Day 11 of 14) Started on: 9/8/19 7:55 AM	5	[Icons]
a3d2b069	Package CppConFuzzTargetVcAsan	Windows Server 2019 Datacenter x64	9/7/19 11:46 PM	Stopped	5	[Icons]

Azure MSRD service

Contact us Privacy & cookies Terms of use Trademarks Third Party Notices © Microsoft 2019

{ ASan + Fuzzing }



<https://sched.co/e7C0>



I hope you're now as excited
as I am for leveraging the power
of ASan on Windows

Looking forward to many days of bug-fixing ahead 🤨

Process: [20684] advinst.exe Thread: [17432] Main Thread Stack Frame: ATL::CWindowImplBaseT<ATL::CWindow,A

```
3602 template <class TBase, class TwinTraits> <T> Provide sample template arguments for IntelliSense
3603 HWND CWindowImplBaseT< TBase, TwinTraits >::Create(
3604     _In_opt_ HWND hWndParent,
3605     _In_ _U_RECT rect,
3606     _In_z_ LPCTSTR szWindowName,
3607     _In_ DWORD dwStyle,
3608     _In_ DWORD dwExStyle,
3609     _In_ _U_MENUorID MenuOrID,
3610     _In_ ATOM atom,
3611     _In_opt_ LPVOID lpCreateParam)
3612 {
3613     ATLASSUME(this->m_hWnd == NULL);
3614
3615     // Allocate the thunk structure here, where we can fail gracefully.
3616     BOOL result = this->m_thunk.Init(NULL, NULL);
3617     if (result == FALSE) {
3618         SetLastError(ERROR_OUTOFMEMORY);
3619         return NULL;
3620     }
3621
3622     if(atom == 0)
3623         return NULL;
```

Diagnostic Tools: Diagnostics session: 10 seconds (10.7s selected)

CPU (% of all processors)

Summary Events Memory Usage CPU Usage

Events: Show Events (1 of 1)

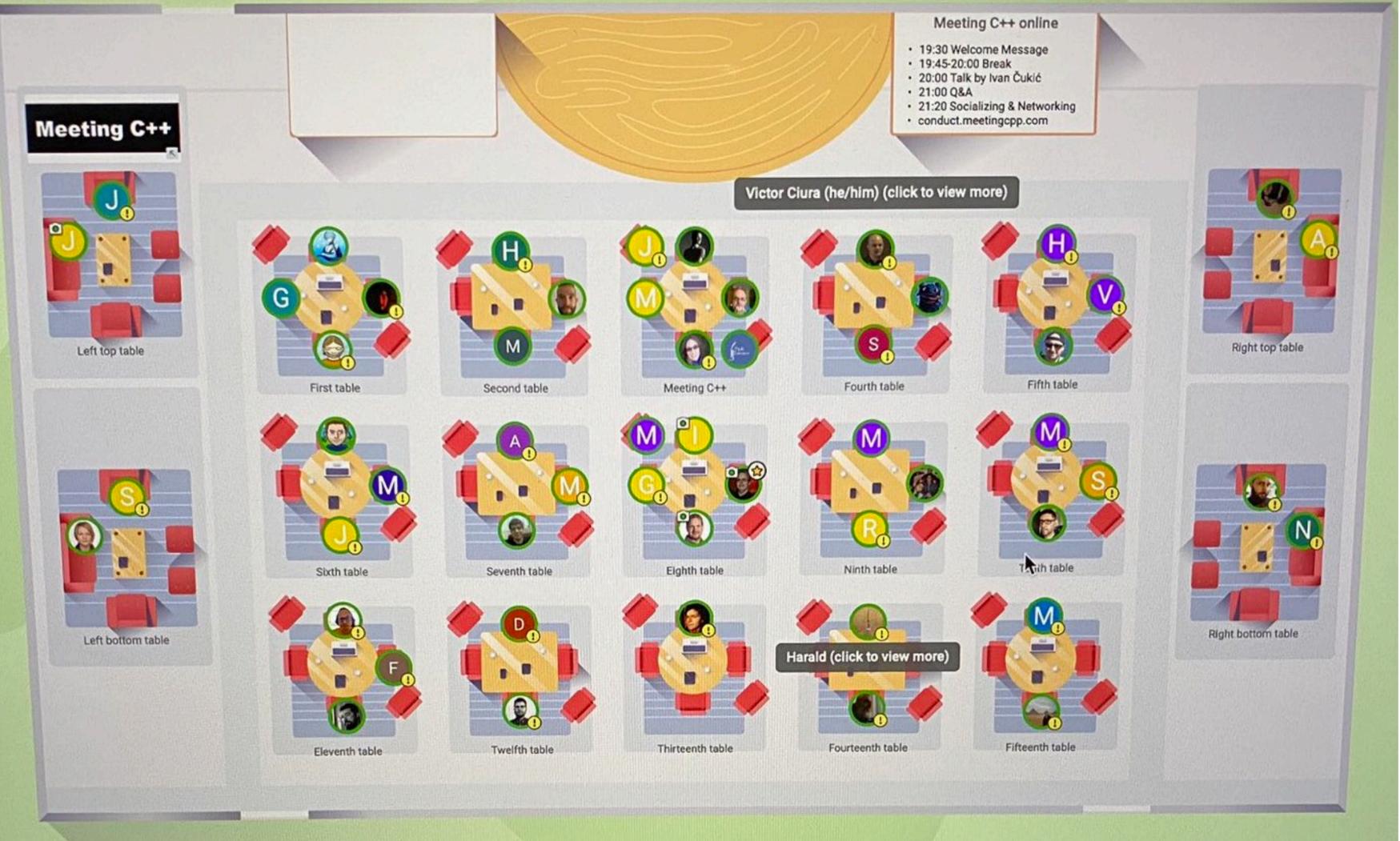
Memory Usage: Take Snapshot, Enable heap profiling (affects performance)

Locals:

Name	Value	Type
this	0x146ecad4 {m_TextSize=...	ATL::C...
atom	50057	unsign...
dwExStyle	0	unsign...
dwStyle	1442840576	unsign...
hWnd	0xc0000000 {unused=???	HWND...
hWndPar...	0x009d004a {Inside advin...	HWND...
lpCreateP...	0x00000000	void *
MenuOrID	{m_hMenu=0x00000000 ...	ATL::_U...
rect	{m_lpRect=0x08c1c004 {a...	ATL::_U...
result	0	int
szWindow...	0x00000000 <NULL>	const ...

Call Stack:

- advinst.exe!ATL::CWindowImplBaseT<ATL::CWindow,ATL::CWinTraits<1442840576,0>>::Create(HWND__ * hWndParent, ATL::_U_RECT rect, const wchar_t * szWindowName, DWORD dwStyle, DWORD dwExStyle, _U_MENUorID MenuOrID, ATOM atom, LPVOID lpCreateParam) Line 3616
- advinst.exe!ATL::CWindowImpl<ThemedPaneContainer,ATL::CWindow,ATL::CWinTraits<1442840576,0>>::Create(HWND__ * hWndParent, ATL::_U_RECT rect, const wchar_t * szWindowName, DWORD dwStyle, DWORD dwExStyle, _U_MENUorID MenuOrID, ATOM atom, LPVOID lpCreateParam) Line 3616
- advinst.exe!WTL::CPaneContainerImpl<ThemedPaneContainer,ATL::CWindow,ATL::CWinTraits<1442840576,0>>::Create(HWND__ * hWndParent, ATL::_U_RECT rect, const wchar_t * szWindowName, DWORD dwStyle, DWORD dwExStyle, _U_MENUorID MenuOrID, ATOM atom, LPVOID lpCreateParam) Line 3616
- advinst.exe!MsiFilesView::DoCreate() Line 675
- advinst.exe!MsiFilesView::OnCreate(tagCREATESTRUCTW * aCreateStruct) Line 652
- advinst.exe!MsiFilesView::ProcessWindowMessage(HWND__ * hWnd, unsigned int uMsg, unsigned int wParam, long lParam, long & lResult, unsigned int wData) Line 652
- advinst.exe!MsiFilesView::ProcessWindowMessage(HWND__ * hWnd, unsigned int uMsg, unsigned int wParam, long lParam, long & lResult, unsigned int wData) Line 652
- [External Code]
- atlthunk.dll!Frames below may be incorrect and/or missing, no symbols loaded for atlthunk.dll
- advinst.exe!MsiFilesComponent::GetView(HWND__ * pParent, IViewManager & aViewManager) Line 63
- advinst.exe!MainFrame::CreateView(MainFrame::ComponentView & aView, unsigned int aID) Line 5886
- advinst.exe!MainFrame::SelectGui(unsigned int aID) Line 5701



Q

&

A

?

Meeting C++ 2020

ONLINE

September 24

2020: The Year of Sanitizers?

Victor Ciura

Principal Engineer



@ciura_victor



CAPHYON