

**ACCU  
2021**  
VIRTUAL EVENT

**Bloomberg**  
Engineering

**undo**

 **mosaic**  
CONSULTANTS TO FINANCIAL SERVICES

# AddressSanitizer on Windows

**Victor Ciura**



# *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.

Do you think you have  
good unit tests & coverage  
on your project ?

Probably not...

I have yet to find a team  
happy about this topic

But I reckon you have  
at least one component  
that you're pretty confident about

Would you be surprised  
to find out there are obvious bugs/vulnerabilities  
in that well tested component ?

Probably not

ಠ\_ಠ(ಠ\_ಠ)

I bet you'd like to quickly dig up  
something like this:



**heap-buffer-overflow** on address 0x0a2301b4 at pc 0x005b7a35 bp 0x011df078 sp 0x011df06c  
READ of size 5 at 0x0a2301b4 thread T0

```
#0 0x5b7a4d in __asan_wrap_strlen crt\asan\llvm\compiler-rt\lib\sanitizer_common\sanitizer_common_interceptors.inc:365
#1 0x278eeb in ATL::CStringT<char,0>::StringLength MSVC\14.28.29333\atlmfc\include\atlsimpstr.h:726
#2 0x278a35 in ATL::CStringT<char,0>::SetString MSVC\14.28.29333\atlmfc\include\atlsimpstr.h:602
#3 0x274d69 in ATL::CStringT<char,0>::operator= MSVC\14.28.29333\atlmfc\include\atlsimpstr.h:314
#4 0x274d99 in ATL::CStringT<char,ATL::StrTraitATL<char,ATL::ChTraitsCRT<char>>>::operator=
    MSVC\14.28.29333\atlmfc\include\cstringt.h:1315
#5 0x27469c in ATL::CStringT<char,ATL::StrTraitATL<char,ATL::ChTraitsCRT<char>>>::CStringT
    MSVC\14.28.29333\atlmfc\include\cstringt.h:1115
#6 0x27641a in SerValUtil::DecryptString C:\JobAI\advinst\msicomp\serval\SerValUtil.cpp:85
#7 0x3e1660 in TestSerVal C:\JobAI\testunits\serval\SerValTests.cpp:60
#8 0x5880e5 in FunctionTest::Run C:\JobAI\testunits\Tester.cpp:71
#9 0x5889b1 in Tester::RunTest C:\JobAI\testunits\Tester.cpp:186
#10 0x586ddb in Tester::ExecuteCommandLine C:\JobAI\testunits\Tester.cpp:558
#11 0x5798d1 in main C:\JobAI\testunits\comps\TestComponents.cpp:2236
```

0x0a2301b4 is located 0 bytes to the right of 4-byte region [0x0a2301b0,0x0a2301b4)  
allocated by thread T0

Stay with me for this 90 minute infomercial  
and I'll show you how easy it is

# Address Sanitizer on Windows

ACCU  
2021



@ciura\_victor

**Victor Ciura**  
Principal Engineer





Due to the nature of delivery medium & streaming delays, I prefer to take questions at the end.

Q & A



**ACCU  
2021**  
VIRTUAL EVENT

**Bloomberg**  
Engineering

**undo**

**mosaic**  
CONSULTANTS TO FINANCIAL SERVICES

**Keynote:**

**Refactoring Superpowers:**

**Make Your IDE Do Your Work, Faster and More Safely**

**Clare Macrae**

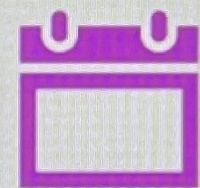
# Safe, How?



IDE



Tests



Version control

# Humans Depend on Tools



**Get to know your tools  
well**



# Programmers Depend on Tools

good code editor  
(or IDE)

linter/formatter

powerful (visual) debugger

automated refactoring tools

build system

package manager

CI/CD service

SCM client

code reviews platform

recent compiler(s)  
[conformant/strict]

perf profiler

test framework

static analyzer

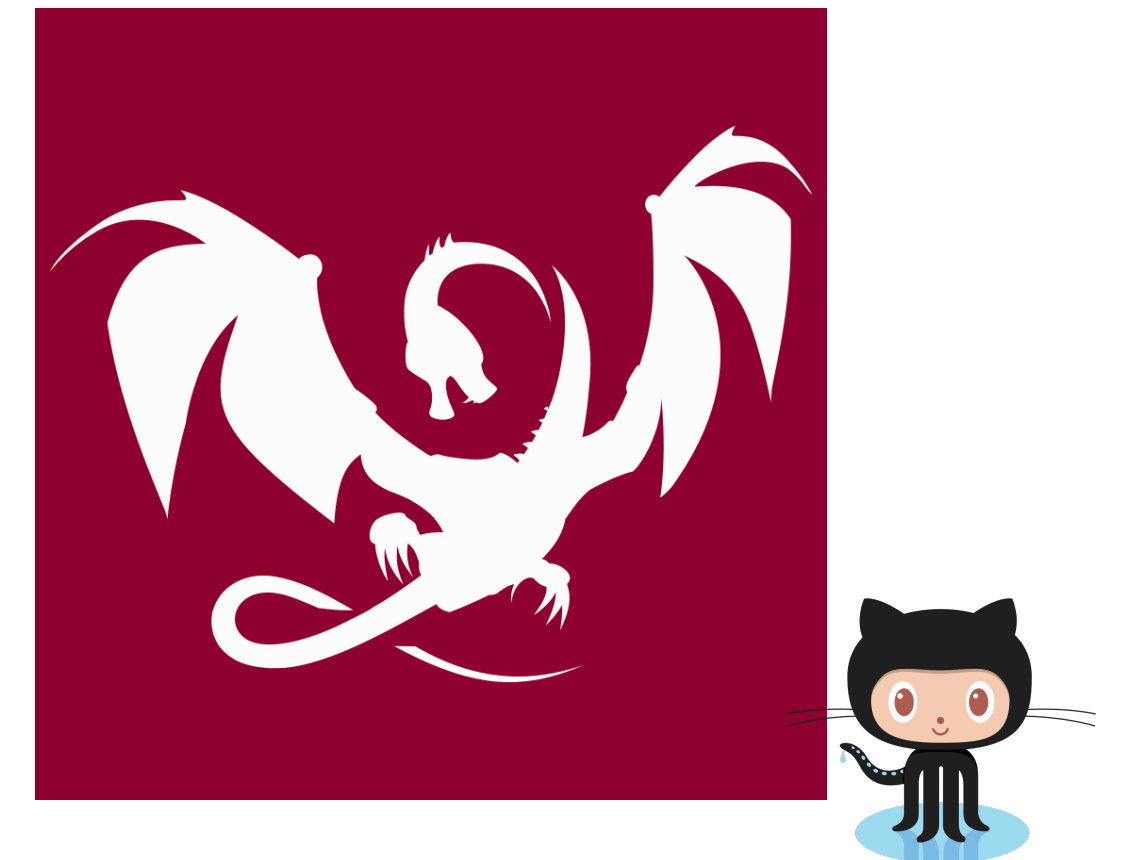
dynamic analyzer  
(runtime)

+ fuzzing

# I'm a tool maker



**Advanced Installer**



**Clang Power Tools**

Free/OSS

 [@ciura\\_victor](https://twitter.com/ciura_victor)

# Vignette in 3 parts

Static Analysis

Dynamic Analysis

Warm Fuzzy Feelings

# Part I

# Static Analysis



# C++ Core Guidelines Checker



[docs.microsoft.com/en-us/cpp/code-quality/quick-start-code-analysis-for-c-cpp](https://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](https://docs.microsoft.com/en-us/cpp/code-quality/code-analysis-for-cpp-corecheck)

[devblogs.microsoft.com/cppblog/new-safety-rules-in-c-core-check/](https://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
<b>C++ Core Check Arithmetic Rules</b>	These rules enforce checks related to <a href="#">arithmetic operations</a> from the <a href="#">C++ Core Guidelines</a> .
<b>C++ Core Check Bounds Rules</b>	These rules enforce the <a href="#">Bounds profile</a> of the <a href="#">C++ Core Guidelines</a> .
<b>C++ Core Check Class Rules</b>	These rules enforce checks related to <a href="#">classes</a> from the <a href="#">C++ Core Guidelines</a> .
<b>C++ Core Check Concurrency Rules</b>	These rules enforce checks related to <a href="#">concurrency</a> from the <a href="#">C++ Core Guidelines</a> .
<b>C++ Core Check Const Rules</b>	These rules enforce <a href="#">const-related checks</a> from the <a href="#">C++ Core Guidelines</a> .
<b>C++ Core Check Declaration Rules</b>	These rules enforce checks related to <a href="#">declarations</a> from the <a href="#">C++ Core Guidelines</a> .
<b>C++ Core Check Enum Rules</b>	These rules enforce <a href="#">enum-related checks</a> from the <a href="#">C++ Core Guidelines</a> .
<b>C++ Core Check Experimental Rules</b>	These rules collect some experimental checks. Eventually, we expect these checks to be moved to other rulesets or removed completely.
<b>C++ Core Check Function Rules</b>	These rules enforce checks related to <a href="#">functions</a> from the <a href="#">C++ Core Guidelines</a> .
<b>C++ Core Check GSL Rules</b>	These rules enforce checks related to the <a href="#">Guidelines Support Library</a> from the <a href="#">C++ Core Guidelines</a> .



[docs.microsoft.com/en-us/cpp/code-quality/code-analysis-for-cpp-corecheck](https://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

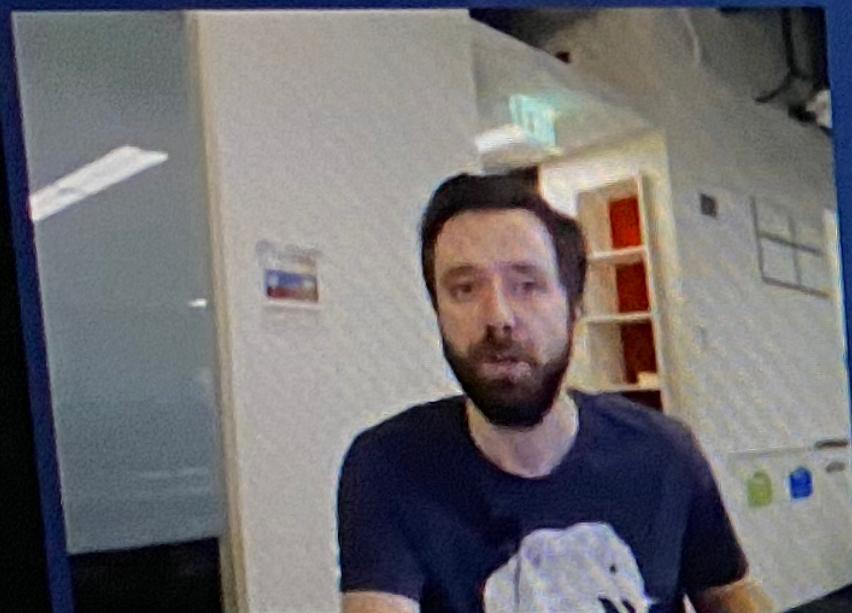
- Missing default label in switch statements
- Unannotated fall through in switch statements
- Expensive range-for copy
- Expensive copy with the auto keyword



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](http://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- 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.	<input type="checkbox"/>	Off
bugprone- 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.	<input type="checkbox"/>	Off
bugprone- 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.	<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





# 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

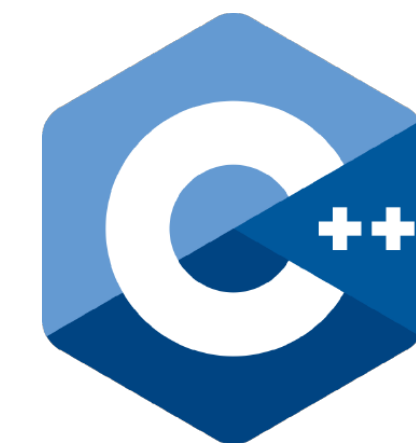
A semicolon-separated list of class names that should be treated as handles. By default only `std::string_view` is considered.

<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>

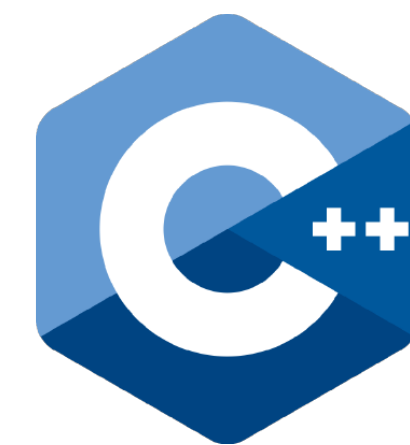
# Lifetime profile v1.0

## 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>

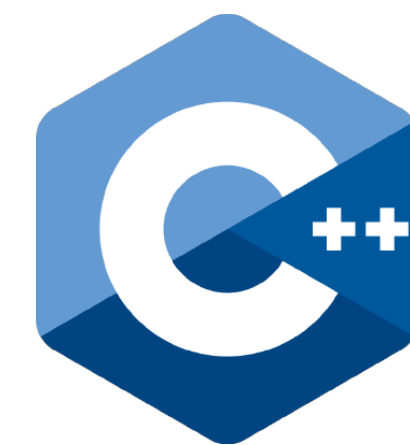
# Lifetime profile v1.0

## 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**

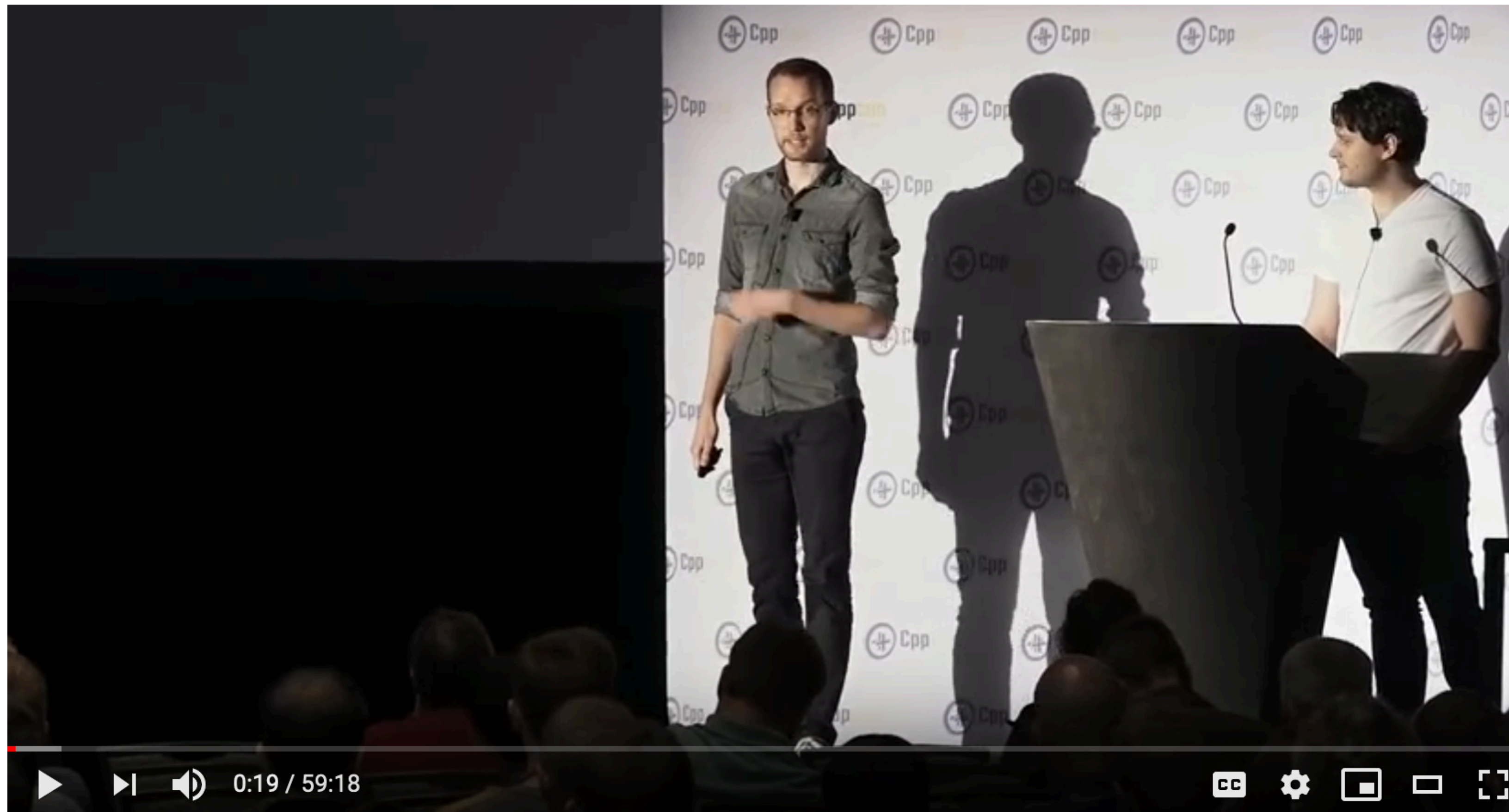
**warning:** initializing pointer member to point to a temporary object whose lifetime is shorter than the lifetime of the constructed object

```
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 Visual Studio Settings window with the 'Tidy' tab selected. The 'Use checks from' dropdown is set to 'CustomChecks'. The 'Predefined Checks' button is 'Select'. 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' button is 'Export'.

Use checks from	CustomChecks
Predefined Checks	Select
Custom Checks	modernize-* ← your <i>custom</i> checks
Header filter	.*
Custom executable	C:\dev\llvm\bin\clang-tidy.exe ← your <i>custom</i> clang-tidy build
Format after Tidy	<input checked="" type="checkbox"/>
Tidy on save	<input type="checkbox"/>
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](https://www.youtube.com/watch?v=Iz4C29yul2U)

2021 Victor Ciura | @ciura\_victor - Address Sanitizer on Windows

38



# 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 since **v16.2**

## Clang/LLVM support for MSBuild & CMake Projects

**Ships with Clang (as optional component)**

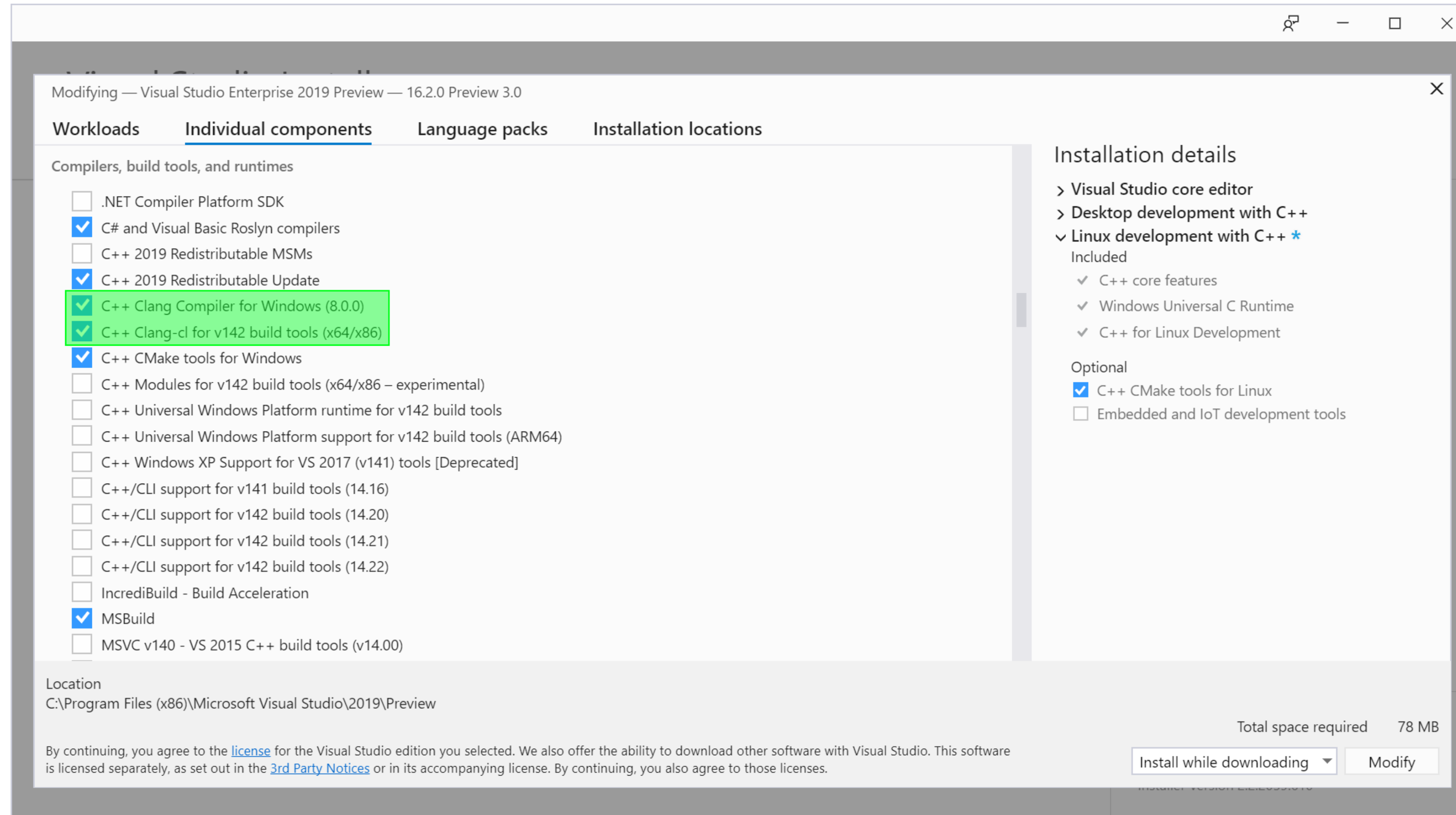
`clang-cl.exe`



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

# Visual Studio 2019

## since v16.2





# Visual Studio 2019

## v16.9

Modifying — Visual Studio Professional 2019 — 16.9.0

Workloads Individual components Language packs Installation locations

clang

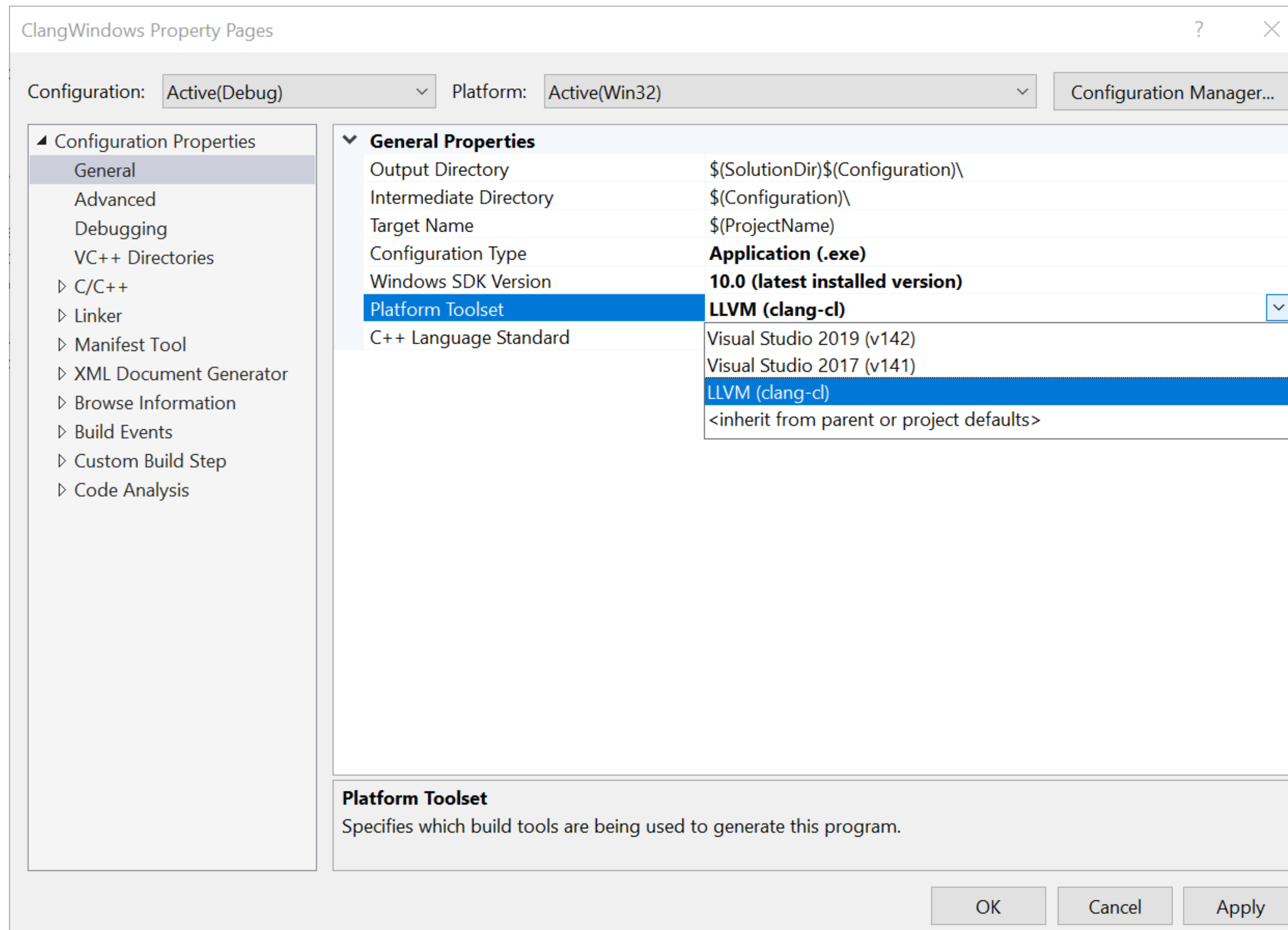
Compilers, build tools, and runtimes

- C++ Clang Compiler for Windows (11.0.0)
- C++ Clang-cl for v142 build tools (x64/x86)



# Visual Studio 2019

## since v16.2



clang-cl.exe

# Visual Studio 2019

since **v16.4**

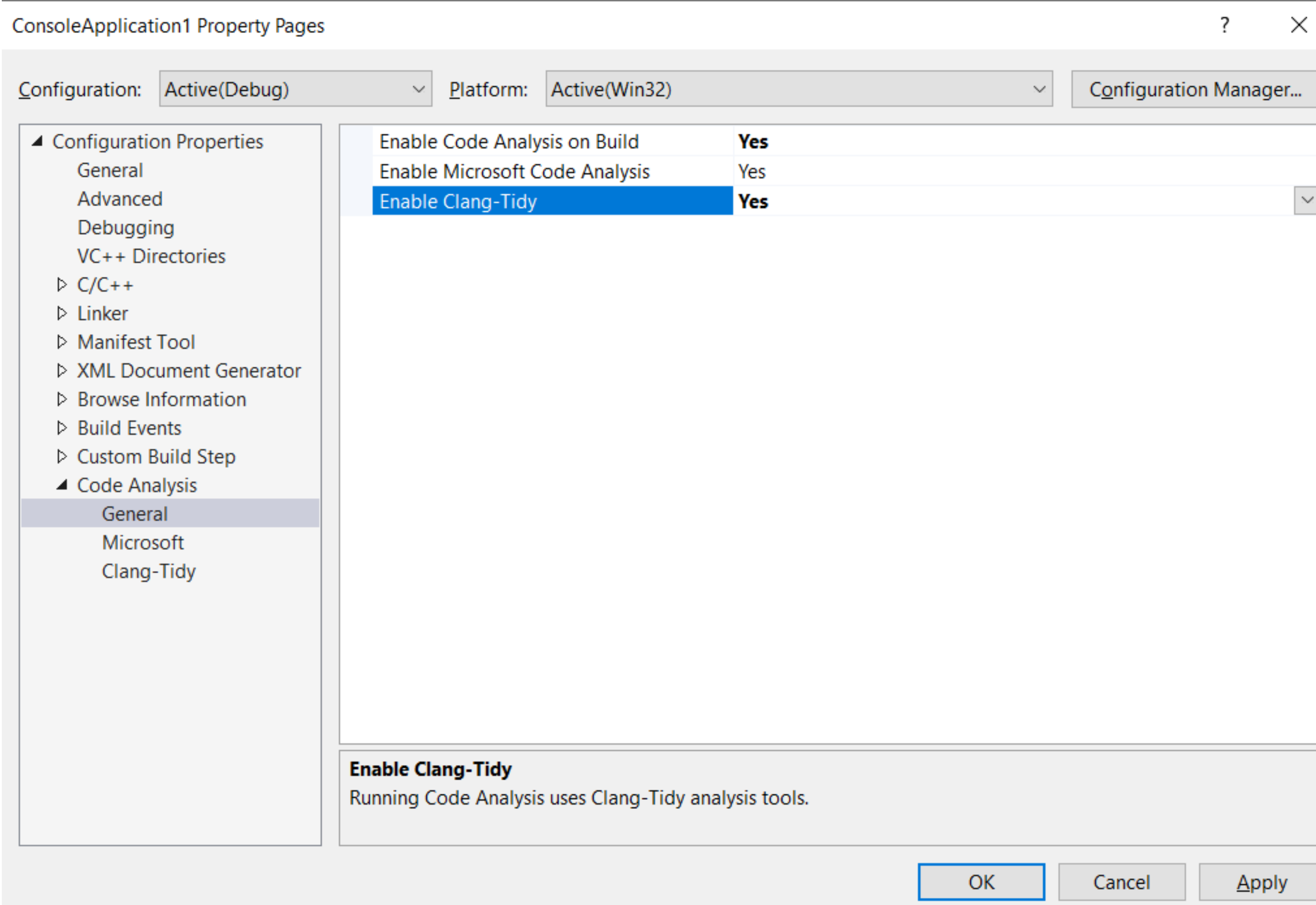
clang-tidy  
code analysis



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

# Visual Studio 2019

## since 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

## since v16.4

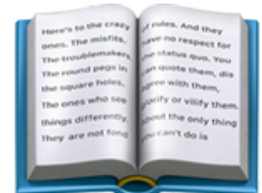
### clang-tidy warnings

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

## since 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**



=



->



Free/OSS

Clang Power Tools

[www.clangpowertools.com](http://www.clangpowertools.com)

LLVM

clang-tidy

clang++

clang-format

clang-check/query

Visual Studio

2015 / 2017 / 2019



# Static vs Dynamic Analysis

# Static Analysis

# Static Analysis

- **offline** (out of the normal compilation cycle) => can take longer to process source code

# Static Analysis

- **offline** (out of the normal compilation cycle) => can take longer to process source code
- is intimately linked to the used **programming language**

# 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**

# 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)

# 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)

# 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)



# 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**

# 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)

# 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

# Dynamic Analysis

- sometimes **intrusive**: you need to compile the program in a special mode

# 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**)

# 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

# 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)



# 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

# 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**

# 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)

# 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)

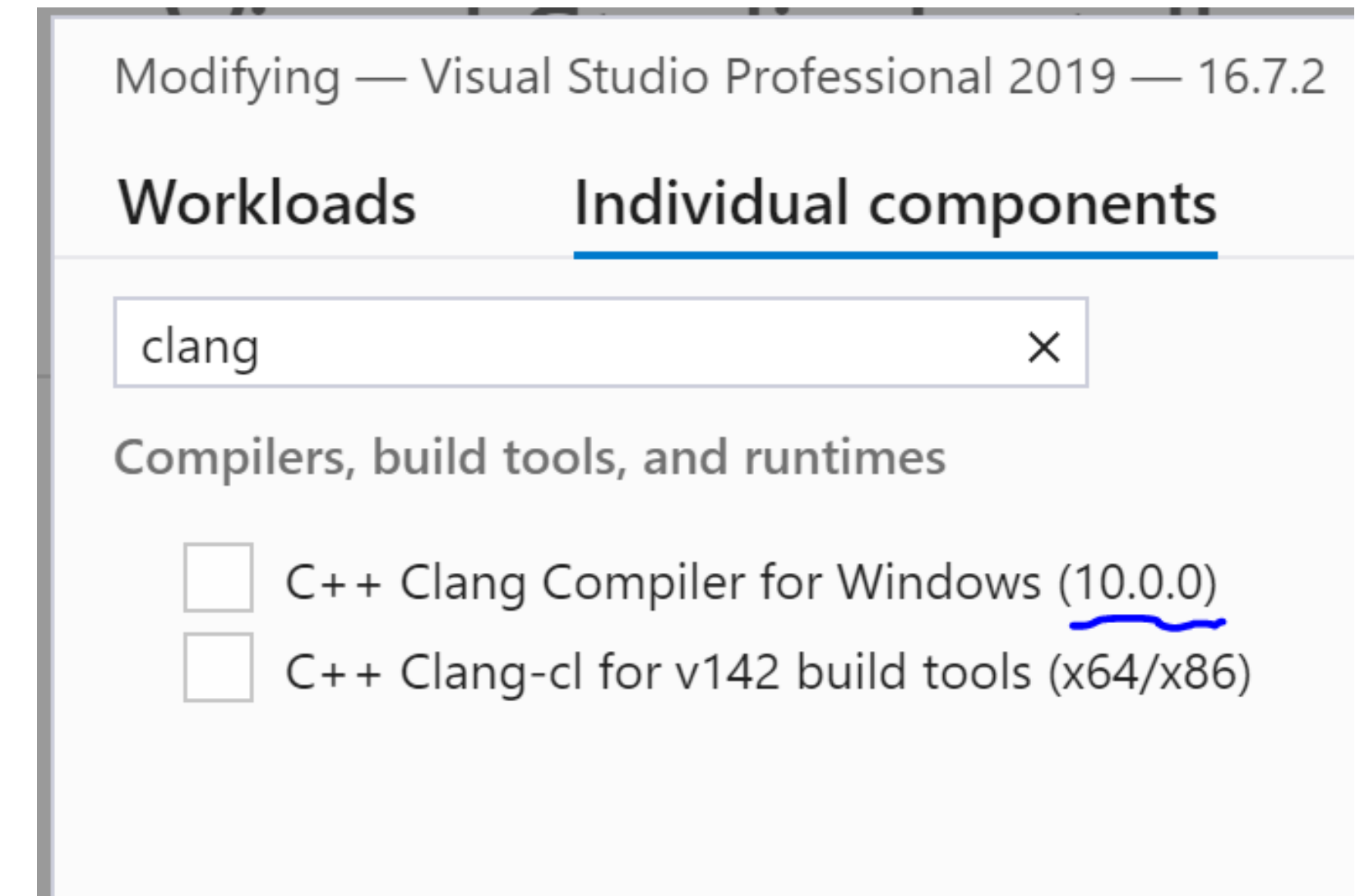
**CFG** is complementary to other exploit mitigations, such as:

- 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

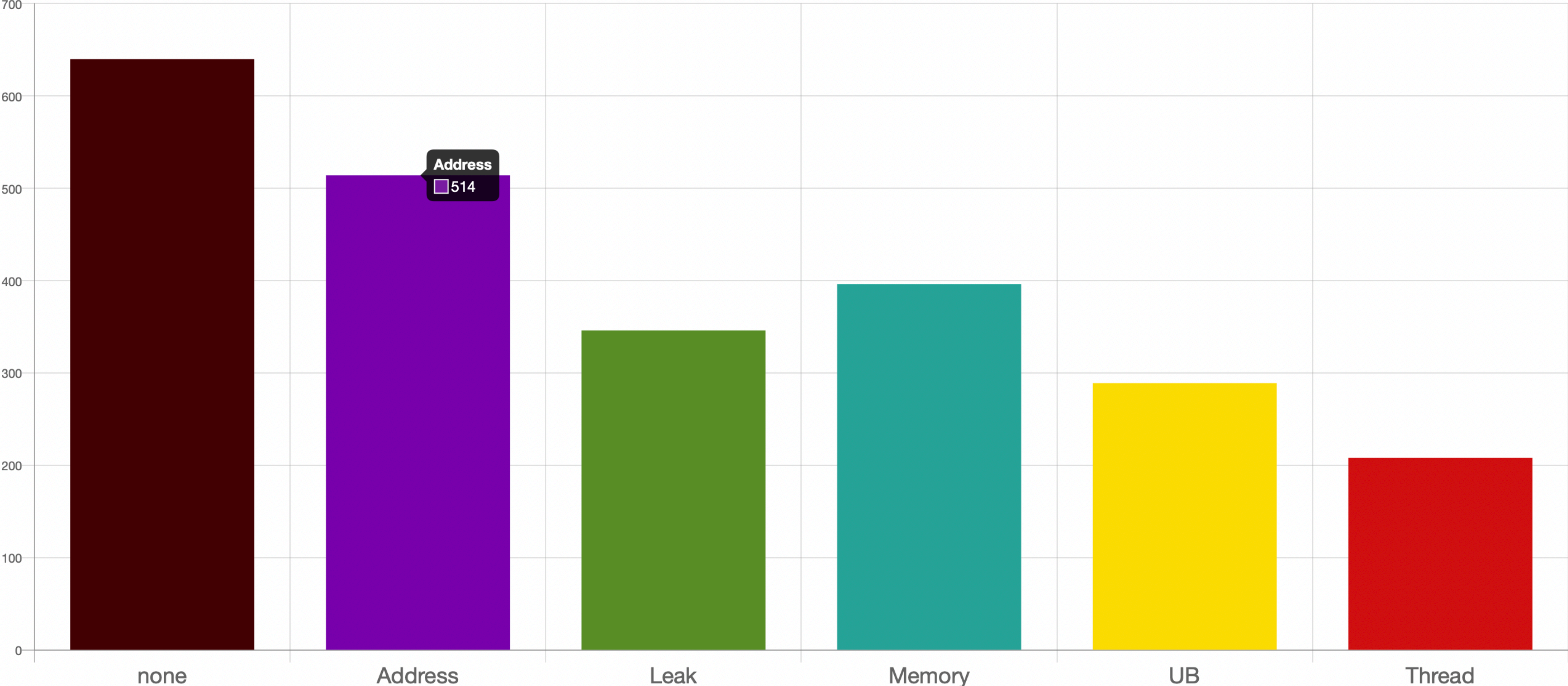
- **AddressSanitizer** - detects addressability issues
- **LeakSanitizer** - detects memory leaks
- **ThreadSanitizer** - detects data races and deadlocks
- **MemorySanitizer** - detects use of uninitialized memory
- **HWASAN** - hardware-assisted AddressSanitizer (consumes less memory)
- **UBSan** -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

[github.com/google/sanitizers](https://github.com/google/sanitizers)

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

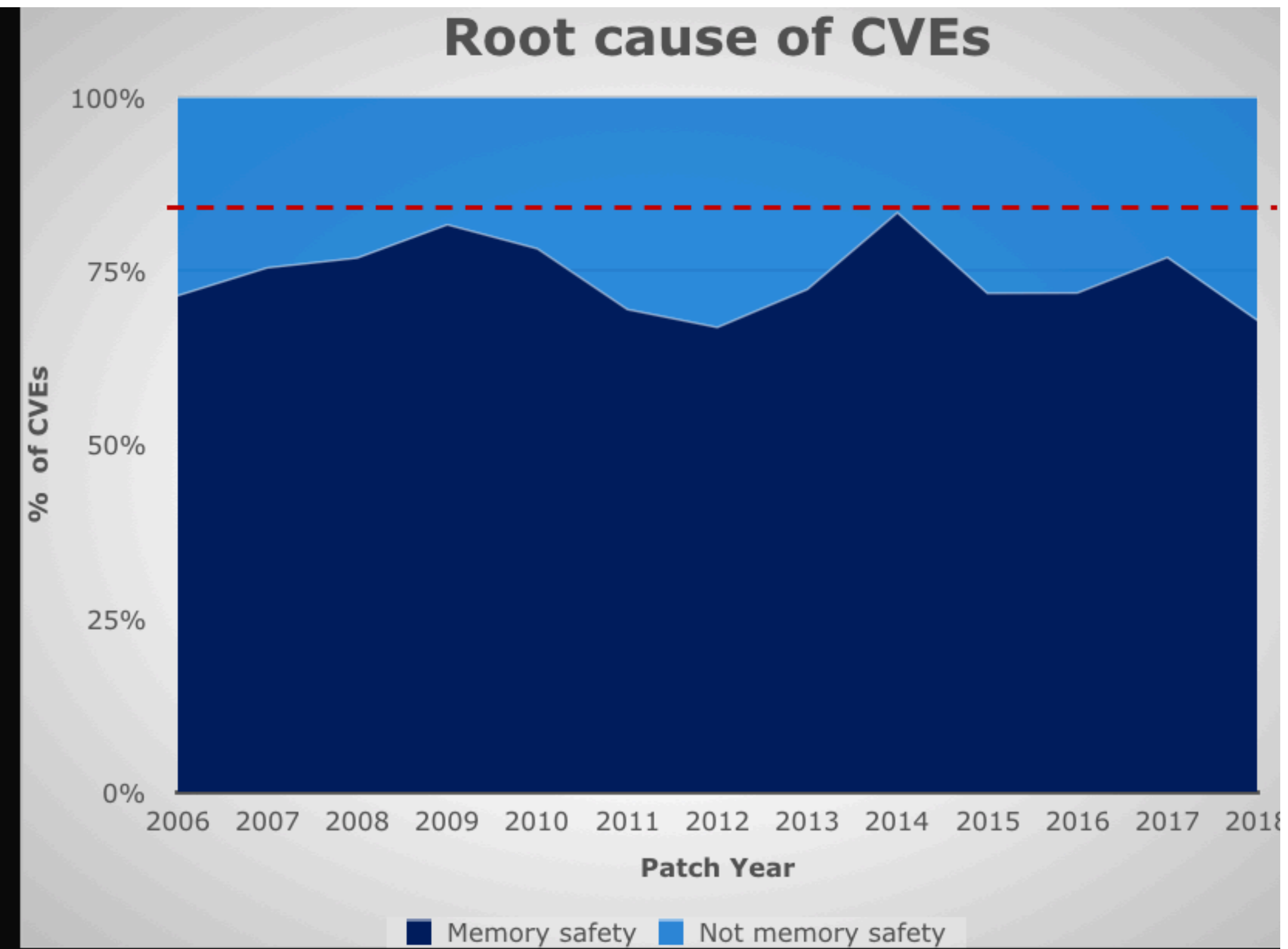
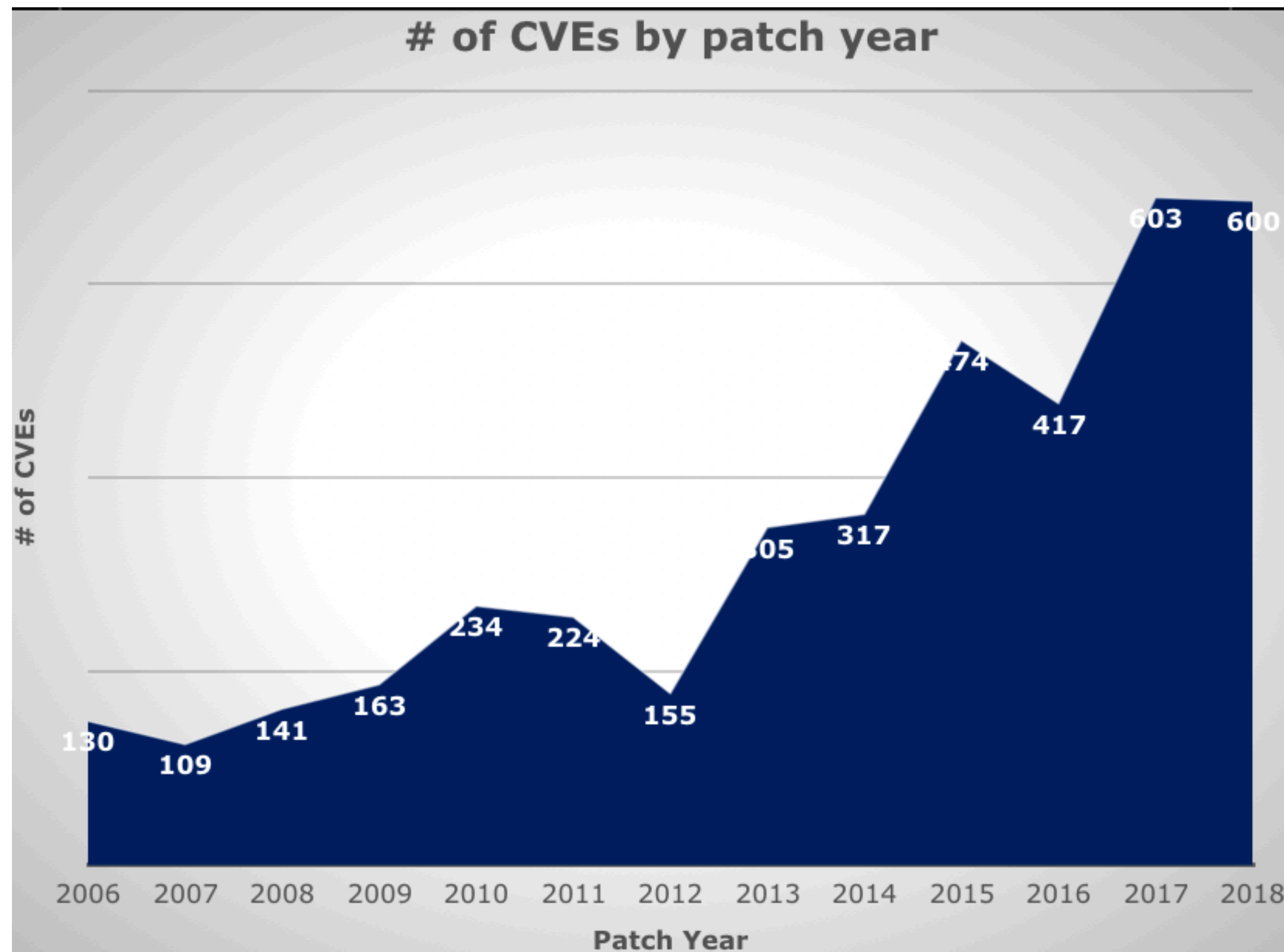
### Meeting C++ Community Survey

Which sanitizers do you use in your builds? (n=1302)



# 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](https://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](https://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](https://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, exp.)

# Visual Studio 2019

since **v16.4**

October 2019

## Address Sanitizer (ASan)



[devblogs.microsoft.com/cppblog/addresssanitizer-asan-for-windows-with-msvc/](https://devblogs.microsoft.com/cppblog/addresssanitizer-asan-for-windows-with-msvc/)

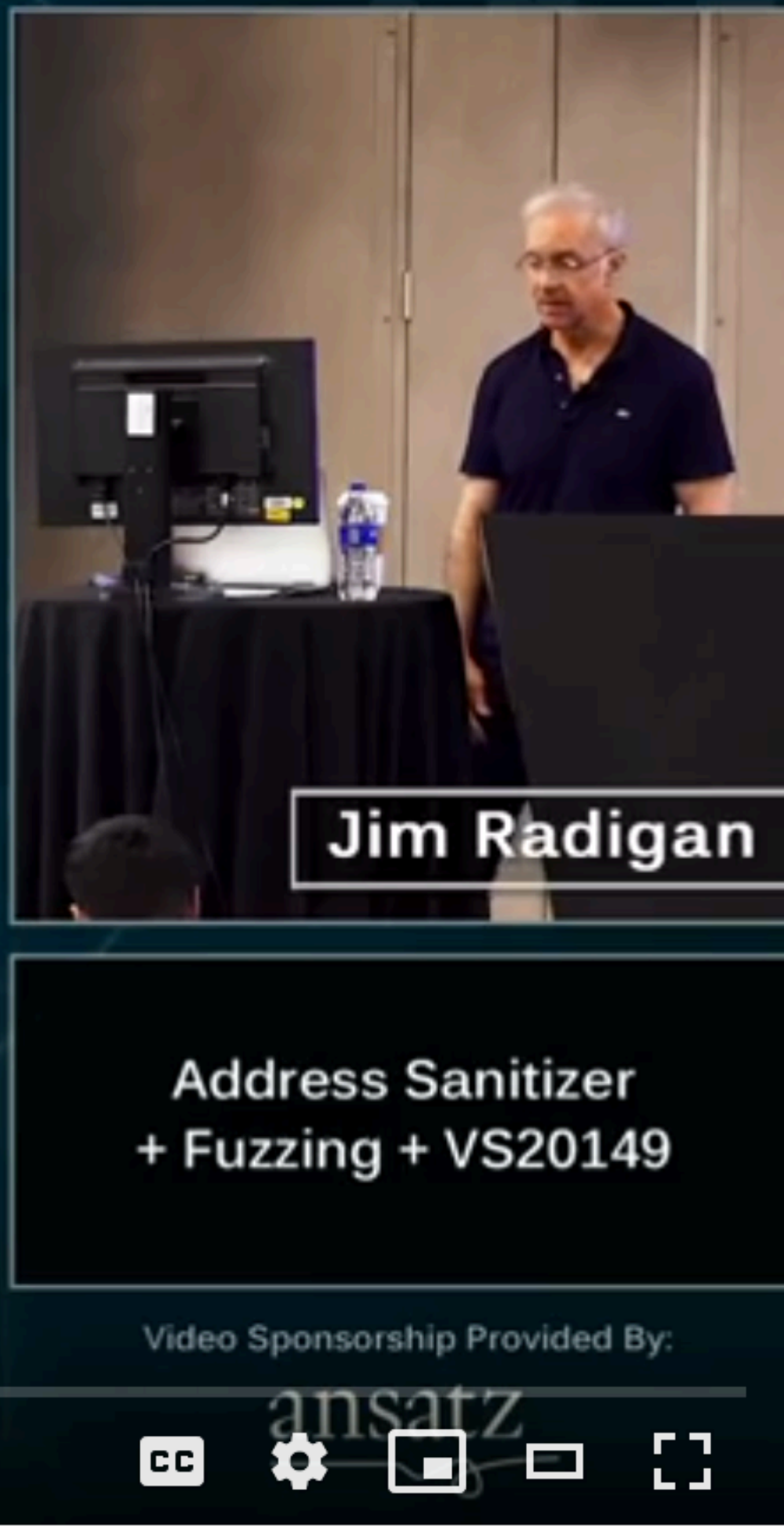





Address Sanitizer + Fuzzing + VS2019

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

## since 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

## since v16.4

Modifying — Visual Studio Professional 2019 — 16.7.2

Workloads Individual components Language packs Installation locations

sanitizer × 

Debugging and testing

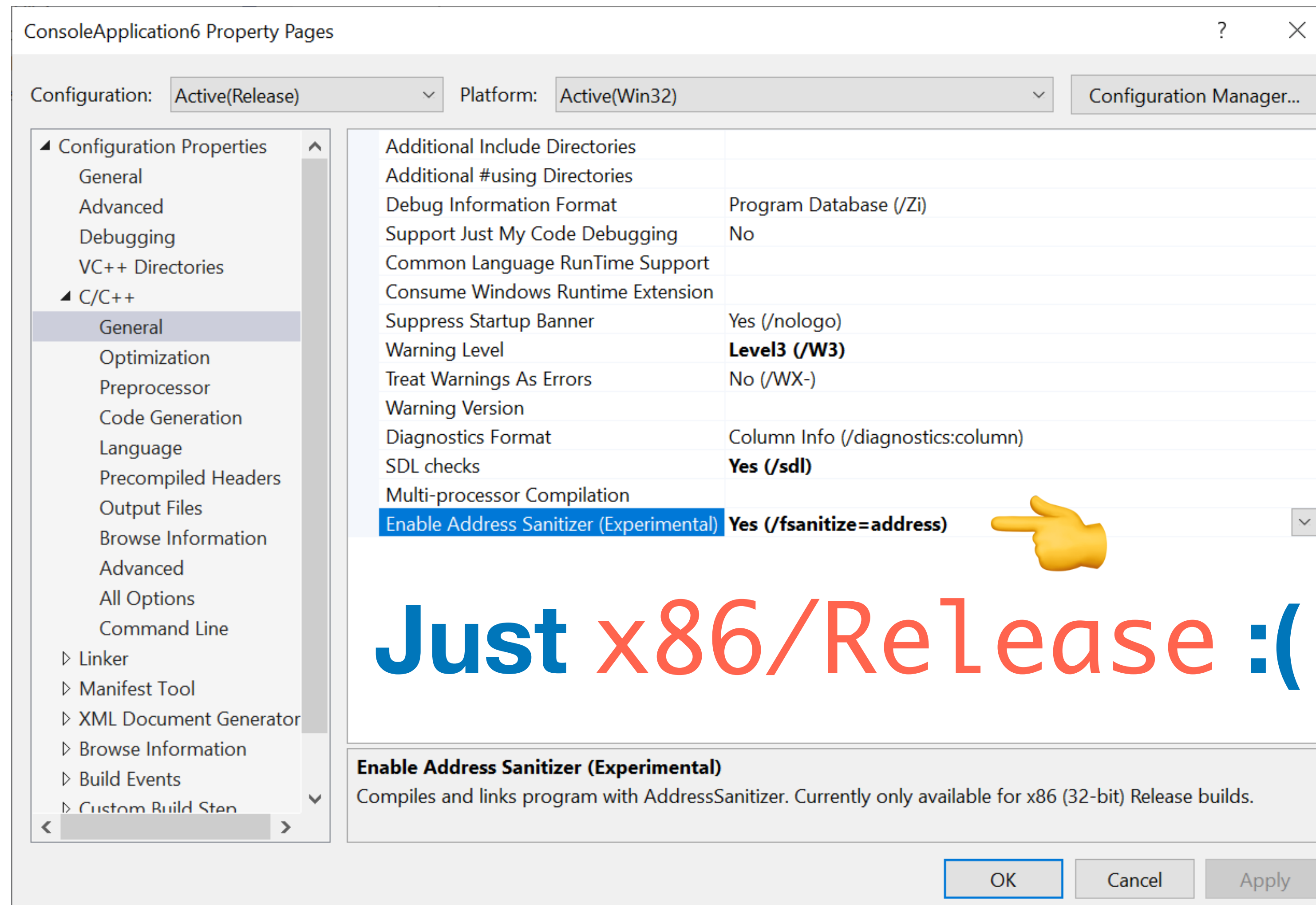
C++ AddressSanitizer (Experimental)

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

# Visual Studio 2019

## since v16.4



**Tech Preview**  
October 2019

# Visual Studio 2019

## since 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	\$(ProjectDir);..\..\.;%(AdditionalIncludeDirectories)
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

**Tech Preview**  
August 2020

# Visual Studio 2019

since **v16.7** August 2020

**+** **x64 & Debug builds**

**support all Debug runtimes: /MTd /MDd**

**Tech Preview**

[docs.microsoft.com/en-us/visualstudio/releases/2019/release-notes#16.7.0](https://docs.microsoft.com/en-us/visualstudio/releases/2019/release-notes#16.7.0)

## **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`

# Visual Studio 2019

## v16.8-9

### New 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`  
(eg. when building from command-line with `/fsanitize:address`)
- `use-after-return (opt-in)`  
(requires code gen that utilizes two stack frames for each function)





March 2021

# Visual Studio 2019

## v16.9



### Available today: Visual Studio 2019 v16.9 and v16.10 Preview

- Address Sanitizer support for Windows
- C++ conformance
- Improved call stack handling
- New memory dump analyzers
- Improvements to GitHub Actions tooling
- .NET productivity enhancements

[Learn what's new](#)

[devblogs.microsoft.com/visualstudio/vs2019-v16-9-and-v16-10-preview-1/](https://devblogs.microsoft.com/visualstudio/vs2019-v16-9-and-v16-10-preview-1/)



March 2021

# Visual Studio 2019

## v16.9

**ASAN is out of Experimental => GA**



[devblogs.microsoft.com/cppblog/address-sanitizer-for-msvc-now-generally-available](https://devblogs.microsoft.com/cppblog/address-sanitizer-for-msvc-now-generally-available)



March 2021

# Visual Studio 2019

v16.9



March 2021

# Visual Studio 2019

## v16.9

- expanded `RtlAllocateHeap` support (fixed compatibility issue with `RtlCreateHeap` and `RtlAllocateHeap` interceptors when creating executable memory pools)



March 2021

# Visual Studio 2019

## v16.9

- expanded `RtlAllocateHeap` support (fixed compatibility issue with `RtlCreateHeap` and `RtlAllocateHeap` interceptors when creating executable memory pools)
- support for the legacy `GlobalAlloc` and `LocalAlloc` family of memory functions  
( `ASAN_OPTIONS=windows_hook_legacy_allocators=true` )



March 2021

# Visual Studio 2019

## v16.9

- expanded `RtlAllocateHeap` support (fixed compatibility issue with `RtlCreateHeap` and `RtlAllocateHeap` interceptors when creating executable memory pools)
- support for the legacy `GlobalAlloc` and `LocalAlloc` family of memory functions  
( `ASAN_OPTIONS=windows_hook_legacy_allocators=true` )
- explicit `error messages` for shadow memory interleaving and interception failure



March 2021

# Visual Studio 2019

## v16.9

- expanded `RtlAllocateHeap` support (fixed compatibility issue with `RtlCreateHeap` and `RtlAllocateHeap` interceptors when creating executable memory pools)
- support for the legacy `GlobalAlloc` and `LocalAlloc` family of memory functions  
( `ASAN_OPTIONS=windows_hook_legacy_allocators=true` )
- explicit `error messages` for shadow memory interleaving and interception failure
- `IDE integration` can now handle the complete collection of `exceptions` which ASan can report



March 2021

# Visual Studio 2019

## v16.9

- expanded `RtlAllocateHeap` support (fixed compatibility issue with `RtlCreateHeap` and `RtlAllocateHeap` interceptors when creating executable memory pools)
- support for the legacy `GlobalAlloc` and `LocalAlloc` family of memory functions  
( `ASAN_OPTIONS=windows_hook_legacy_allocators=true` )
- explicit `error messages` for shadow memory interleaving and interception failure
- `IDE integration` can now handle the complete collection of `exceptions` which ASan can report
- compiler/linker will suggest emitting `debug information` when building with ASan





March 2021



## AddressSanitizer

### AddressSanitizer overview

Build and language reference

Runtime reference

Debugger integration

Shadow bytes

Cloud or distributed testing

### AddressSanitizer error examples

AddressSanitizer error examples

alloc-dealloc-mismatch error

allocation-size-too-big error

calloc-overflow error

double-free error

dynamic-stack-buffer-overflow error

global-buffer-overflow error

heap-buffer-overflow error

heap-use-after-free error


invalid-allocation-alignment error

memcpy-param-overlap error

new-delete-type-mismatch error

stack-buffer-overflow error

# AddressSanitizer

03/05/2021 • 7 minutes to read • 

## Overview

The C & C++ languages are powerful, but can suffer from a class of bugs that affect program correctness and program security. Starting in Visual Studio 2019 version 16.9, the Microsoft C/C++ compiler (MSVC) and IDE supports the *AddressSanitizer*. AddressSanitizer (ASan) is a compiler and runtime technology that exposes many hard-to-find bugs with zero false positives:

- Alloc/dealloc mismatches and new/delete type mismatches
- Allocations too large for the heap
- calloc overflow and alloca overflow
- Double free and use after free
- Global variable overflow
- Heap buffer overflow
- Invalid alignment of aligned values
- memcpy and strcat parameter overlap
- Stack buffer overflow and underflow
- Stack use after return and use after scope
- Memory use after it's poisoned

Use AddressSanitizer to reduce your time spent on:

- Basic correctness
- Cross platform portability
- Security
- Stress testing
- Integrating new code

[docs.microsoft.com/en-us/cpp/sanitizers/asan](https://docs.microsoft.com/en-us/cpp/sanitizers/asan)

# Visual Studio ASan

**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

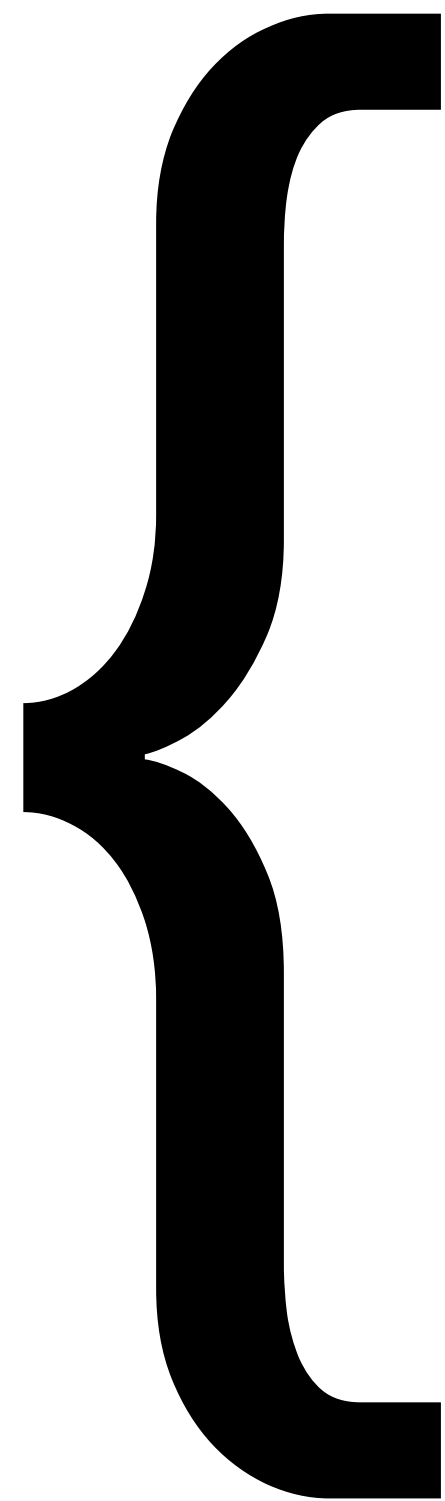
**the surface area of the Microsoft platform is enormous**

**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

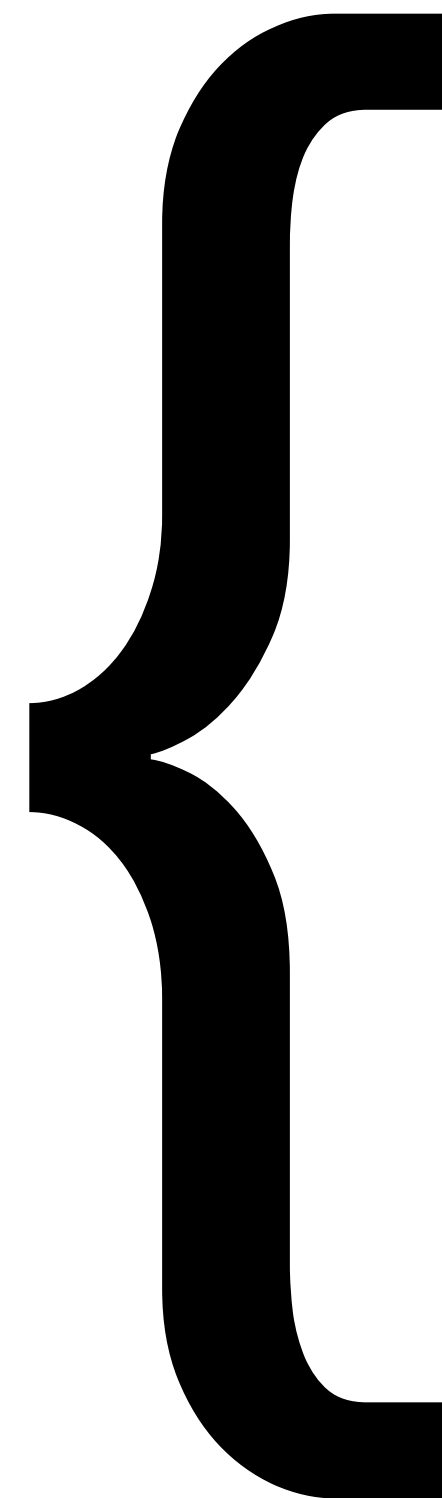
non-standard C++



# Challenges bringing ASan to Windows

the surface area of the Microsoft platform is enormous

non-standard C++



Structured Exception Handling (SEH) /EHa

AV traps 0xc0000005

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

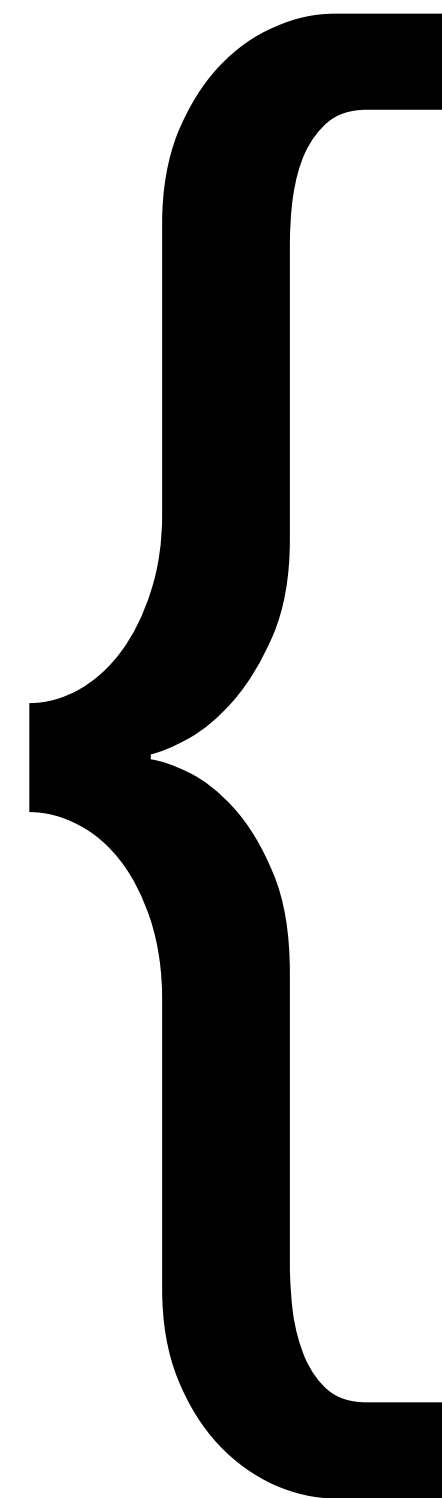
COM

Managed 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)

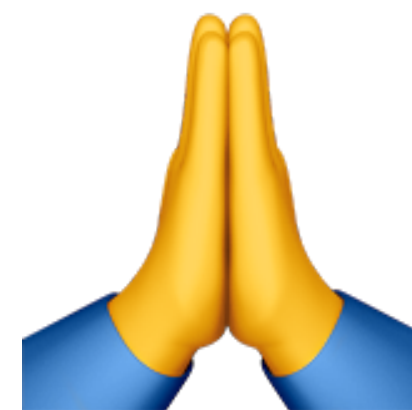
COM

Managed C++

ASan runtime interop with managed code (.NET)

# Visual Studio ASan

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





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

Microsoft is contributing 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](https://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](https://aka.ms/asan)

# Visual Studio ASan CMake

CMakeSettings.json

```
// eg. under the x86-Release configuration  
{  
  "addressSanitizerEnabled": true  
}
```

> build with `/fsanitize:address`

# Address Sanitizer (ASan)

The screenshot shows a Visual Studio IDE window with the following code:

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

The line `array[100] = 1;` is underlined with a red squiggly line, and a red 'X' icon is next to it. An exception dialog box 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...](#)

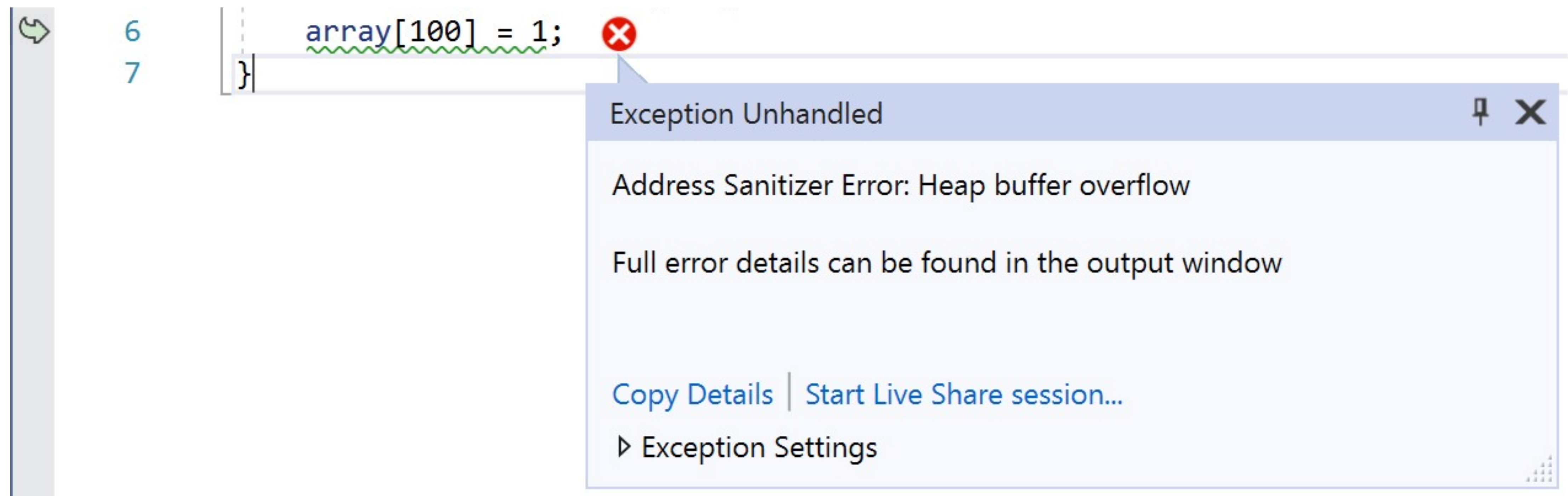
▸ Exception Settings



# 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

**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	

**System Information**

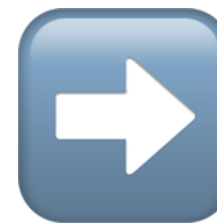
OS Version	10.0.17763
CLR Version(s)	4.6.26702.0

**Modules**

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

**Actions**

- Debug with Managed Only
- Debug with Mixed
- Debug with Native Only
- Debug Managed Memory
- Set symbol paths
- Copy all to clipboard



Exception Unhandled  
ASAN Error: Stack Buffer Overflow

```
109 CloseHandle(fileHandle);  
110  
111 void* freed_pointer = malloc(  
112 free(freed_pointer); //we'll  
113  
114 if (array[0] == 'a') {  
115     if (array[1] == 'b')  
116         if (array[2] == 'c')  
117             if (array[3] ==  
118                 if (array[4]  
119                     if (arr  
120     pri  
121 }  
122  
123 if (array[10] == 'B')  
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);
```

Locals

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

Output

```
0x3019fe0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
0x3019ff0: 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 04[f2]f8 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
```

Visual Studio interface showing a C++ program with a stack buffer overflow. The code in `HeapCorruptionSample.cpp` includes a `main` function that manipulates a `char array[256]`. An `Exception Unhandled` dialog box is open, displaying the error: `ASAN Error: Stack Buffer Overflow`. The dialog also lists several Azure Machine Learning buckets and provides options to view full error details, copy details, or start a collaboration session. A blue arrow points from the dialog to the `Output` window, which shows memory addresses and hex values. The `Locals` window shows the current state of variables: `argc` (2), `argv` (0x04301ad0), `array` (0x00cff6c4), `FileHandle` (0x00000000), `freed_pointer` (0x00000000), and `readBytes` (27).

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
0x05065ed88ff0: 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 fa fa fa fa fa fa fa fa fa fa fa fa fa fa
0x05065ed90050: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
0x05065ed90060: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
0x05065ed90070: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
0x05065ed90080: fa fa fa fa fa fa fa fa fa fa fa fa fa fa 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 fa fa fa fa fa fa fa fa fa fa fa fa fa fa
 0x05065ed90050: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
 0x05065ed90060: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
 0x05065ed90070: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
 0x05065ed90080: fa fa fa fa fa fa fa fa fa fa fa fa fa fa 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 . . .
```



Windows Security

Virus & threat protection

Security scan required

Your administrator requires a security scan of this item. The scan could take up to 10 seconds.

10:32 PM

# 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:

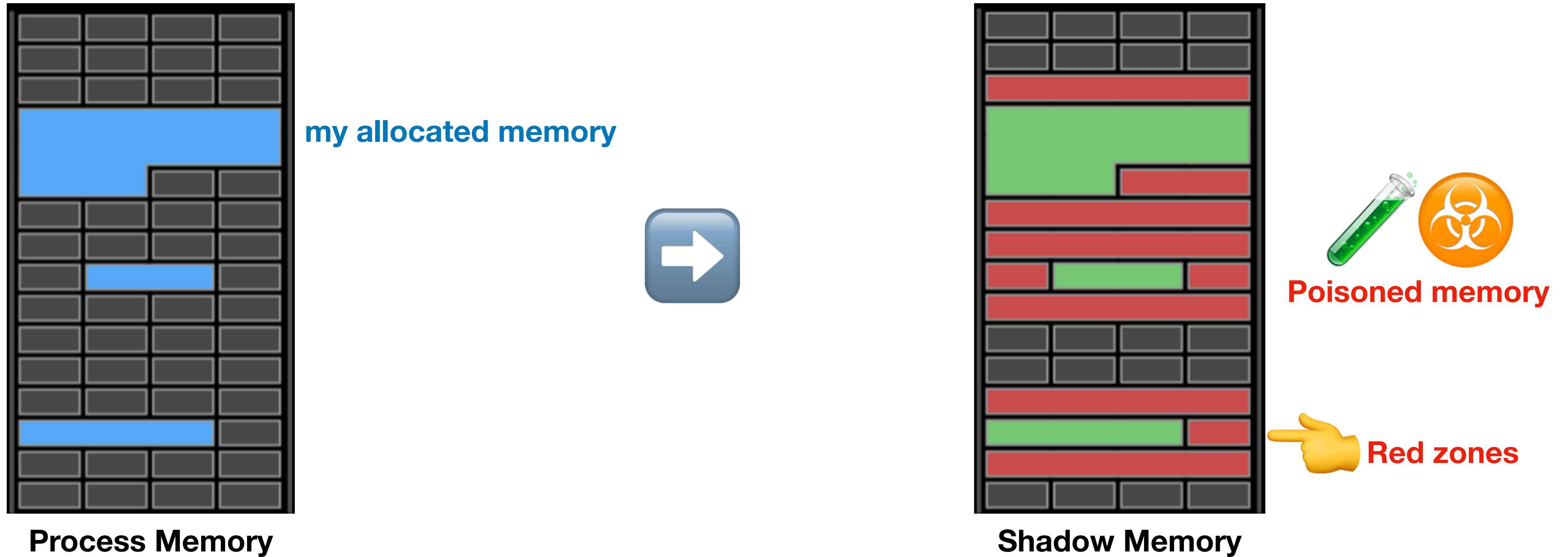
```
0x04d981eef0e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x04d981eef0f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x04d981ef0000: fa fa fa fa fa fa fa fa 00 00 00 00 00 00 00 00
=>0x04d981ef0030: 00 00 00 00 00 00 00 00 00 00 [fa] fa fa fa fa fa
0x04d981ef0040: fa fa fa fa fa fa fa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

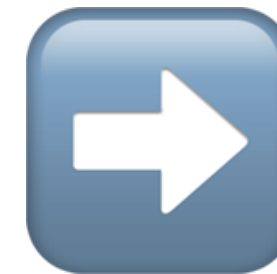
# Shadow Mapping



# 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;  
}
```

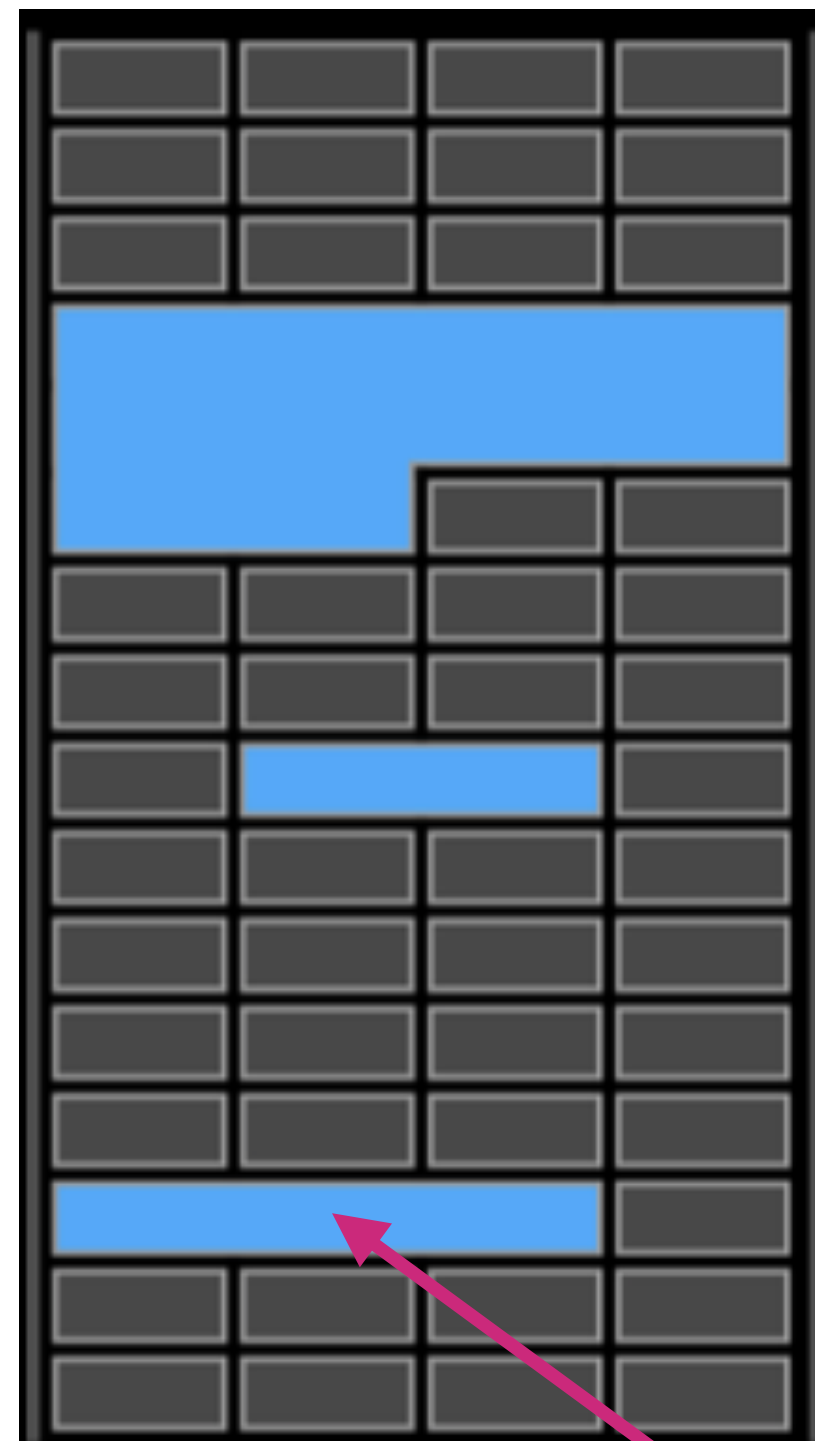
Location of shadow region in memory

A Shadow Byte:

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

Stack use after scope

# Shadow Mapping

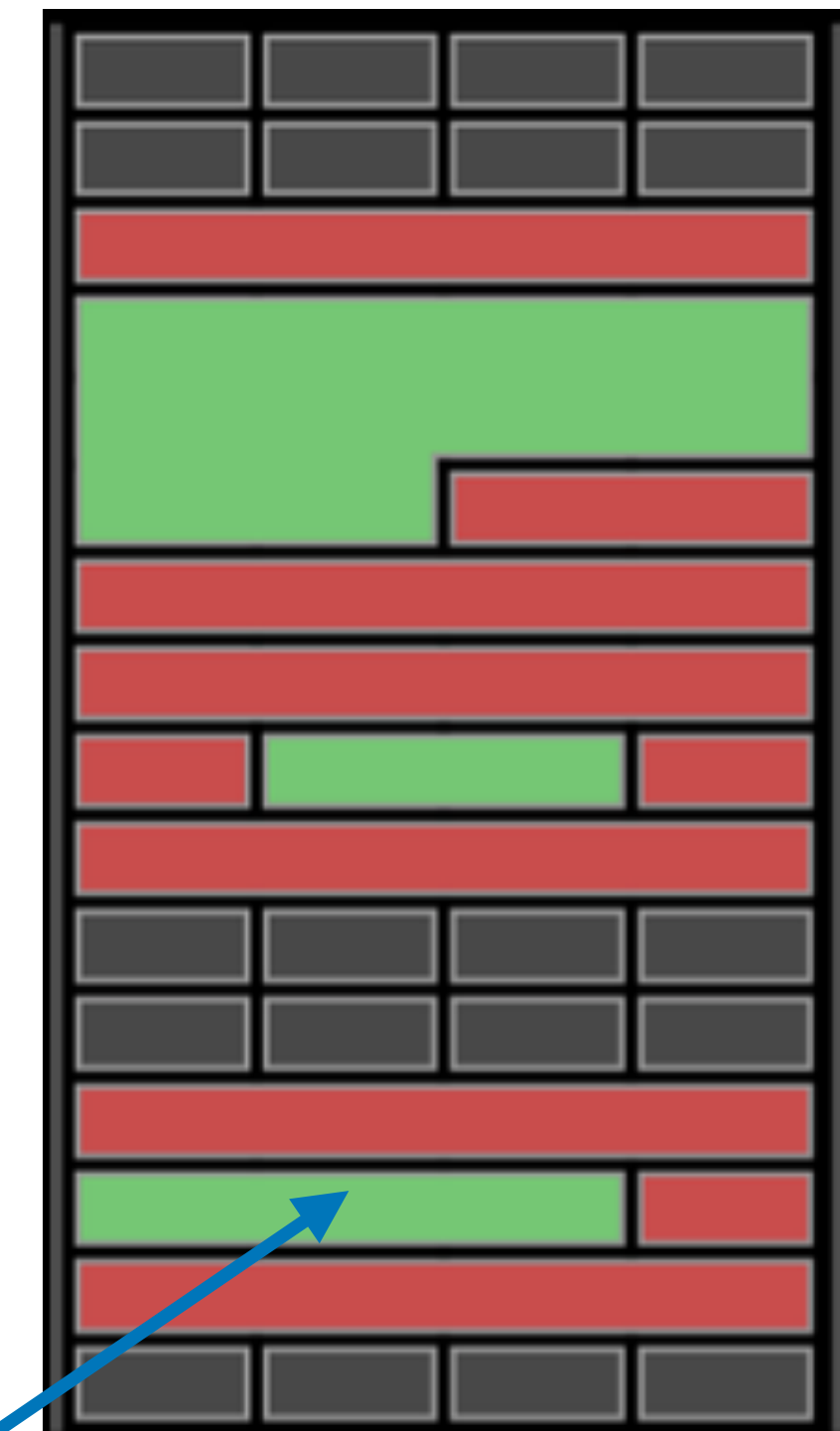


Process Memory

p

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

\*p = 0xf00d

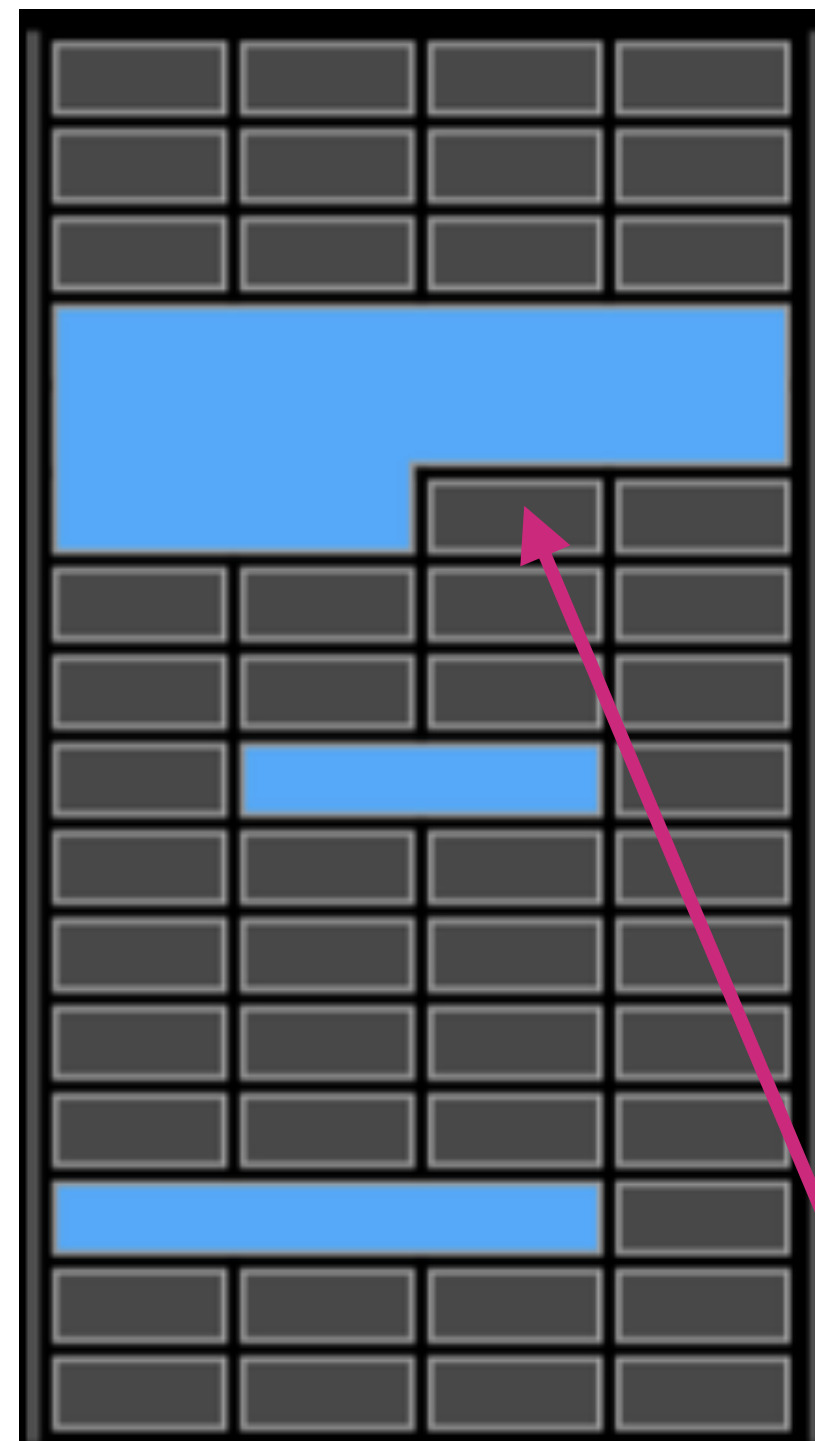


Shadow Memory

ShadowByte(p)



# Shadow Mapping

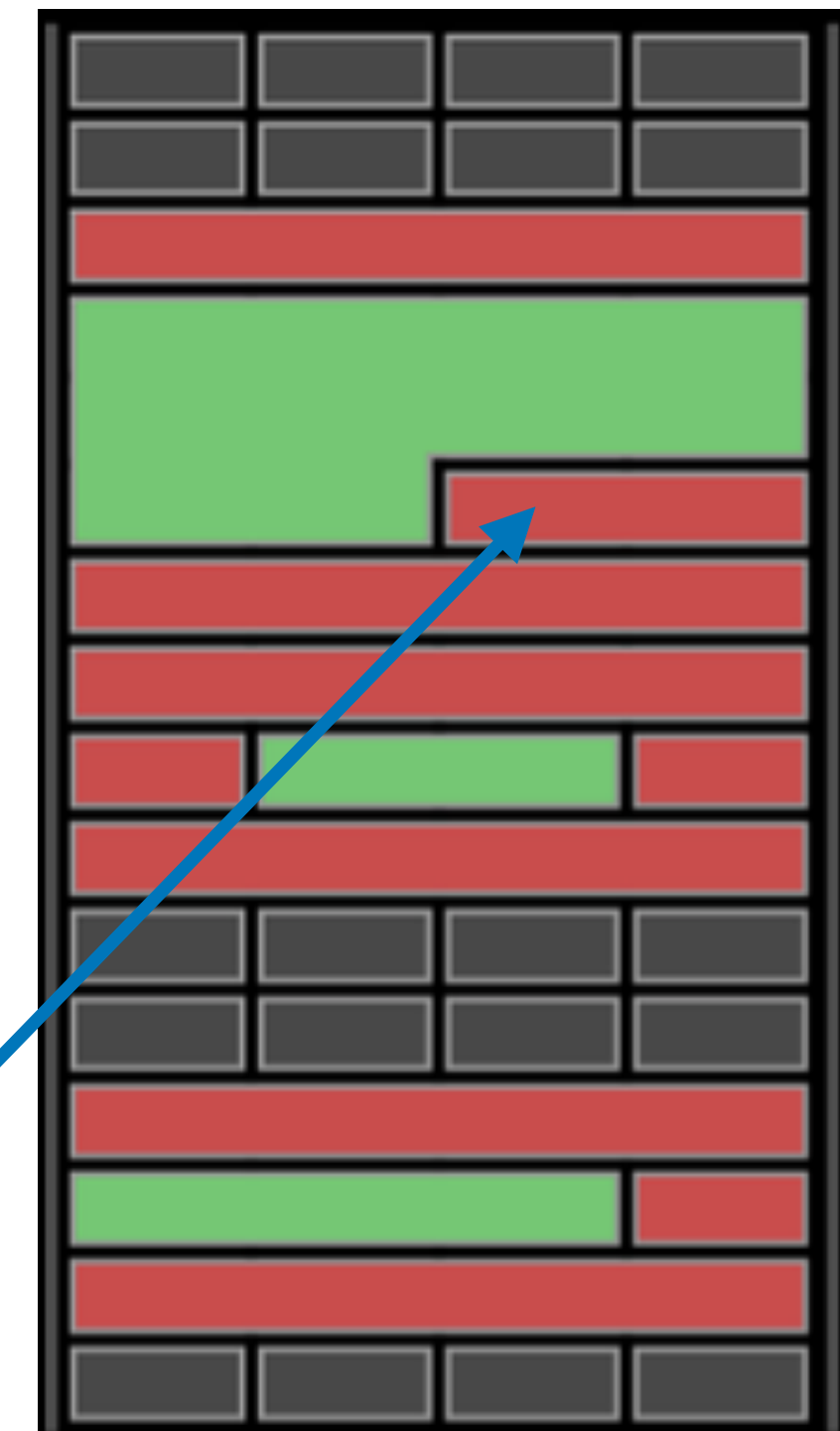


Process Memory

p

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

```
*p = 0xbadf00d
```



Shadow Memory

ShadowByte(p)

# Heap Red Zones

malloc()



ASAN malloc()

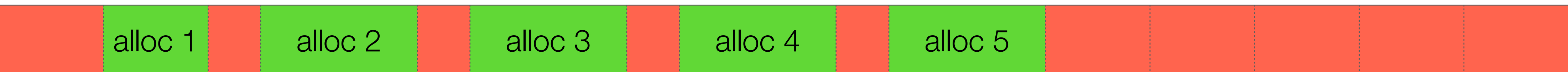


# Heap Red Zones

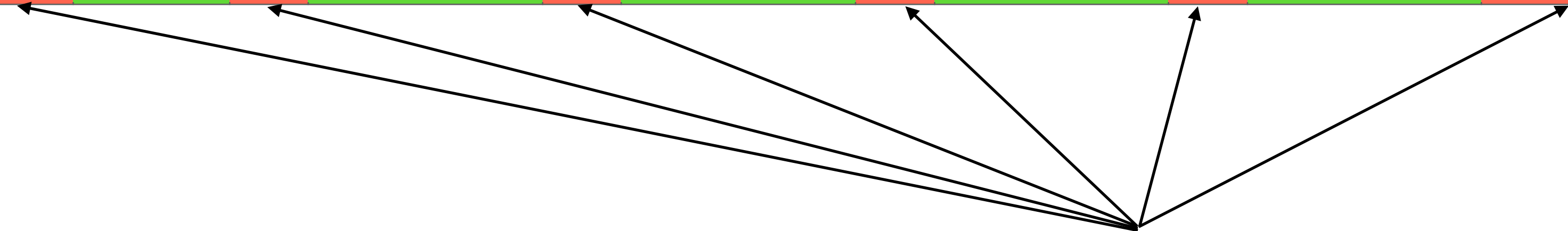
ASAN malloc()



Shadow Memory



Poisoned memory



# Heap Red Zones

ASAN malloc()



When an object is **deallocated**, its corresponding shadow byte is **poisoned** (delays reuse of freed memory)

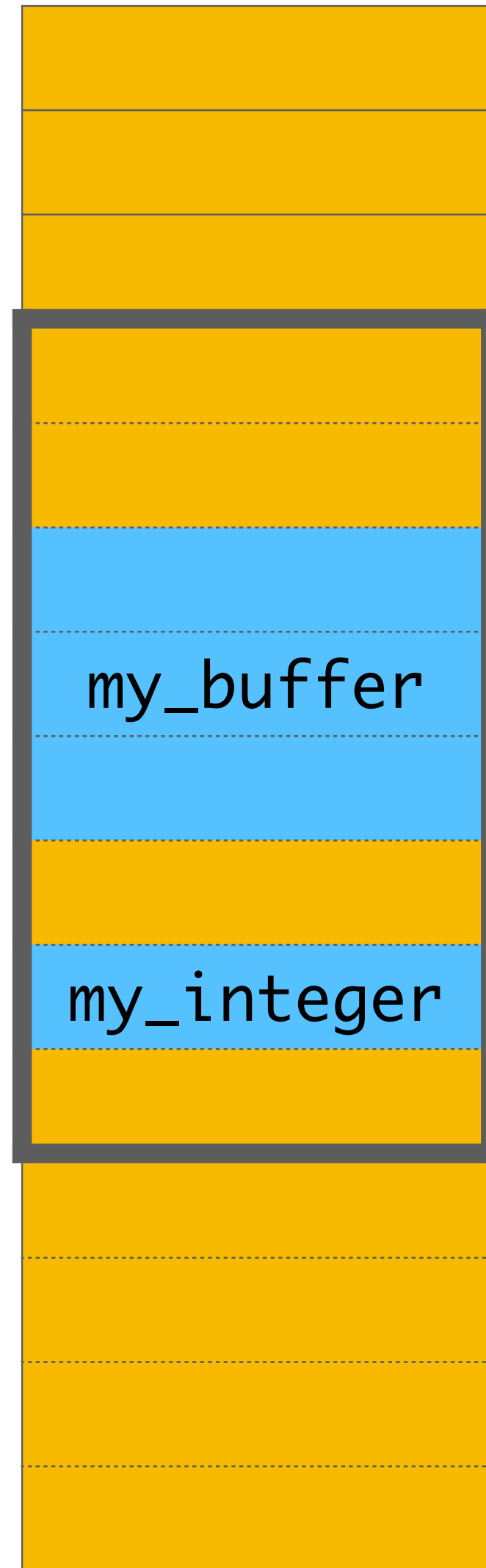
Shadow Memory



Poisoned memory

- Detect:**
- heap underflows/overflows
  - use-after-free & double free

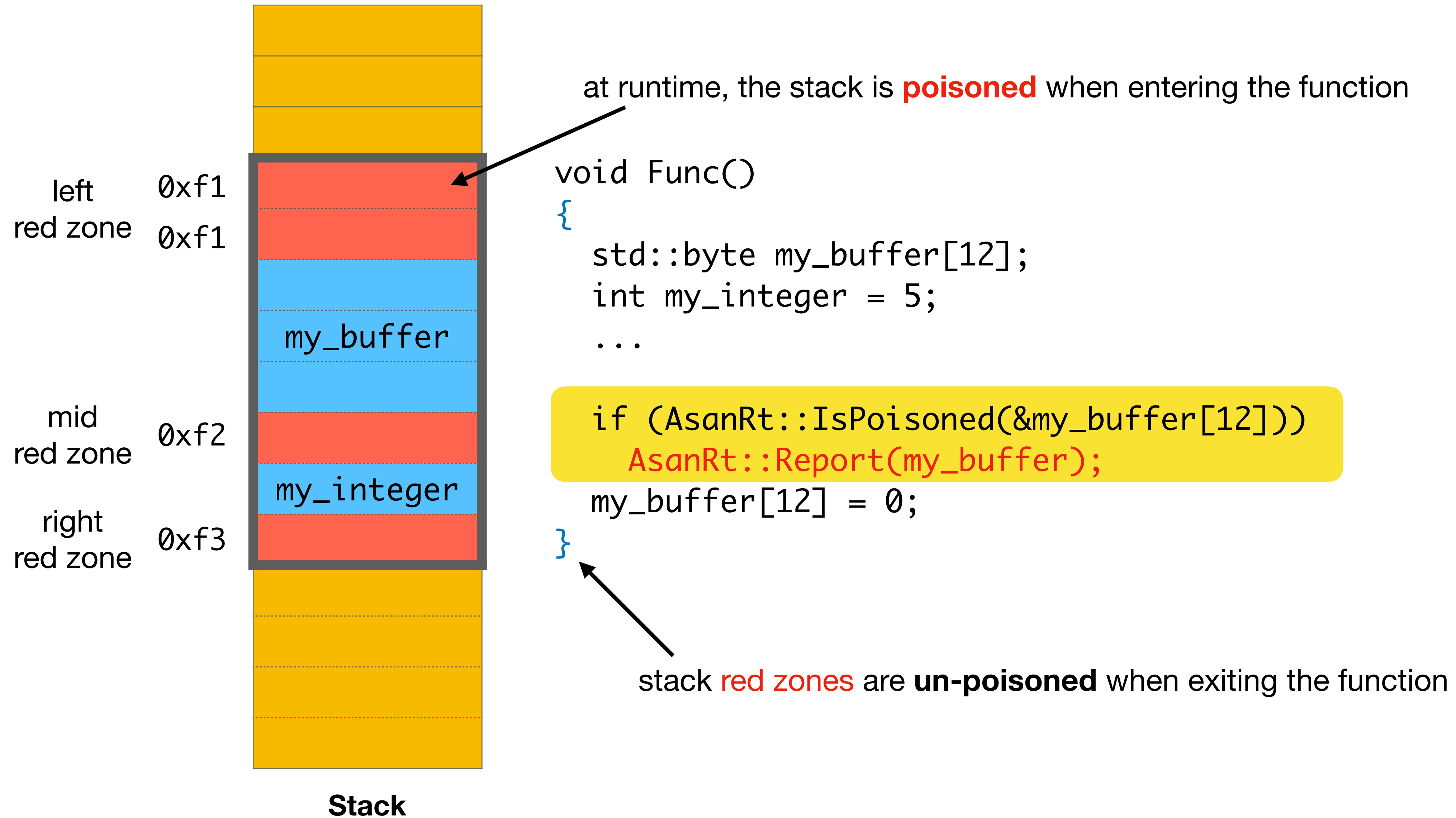
# Stack Red Zones



Stack

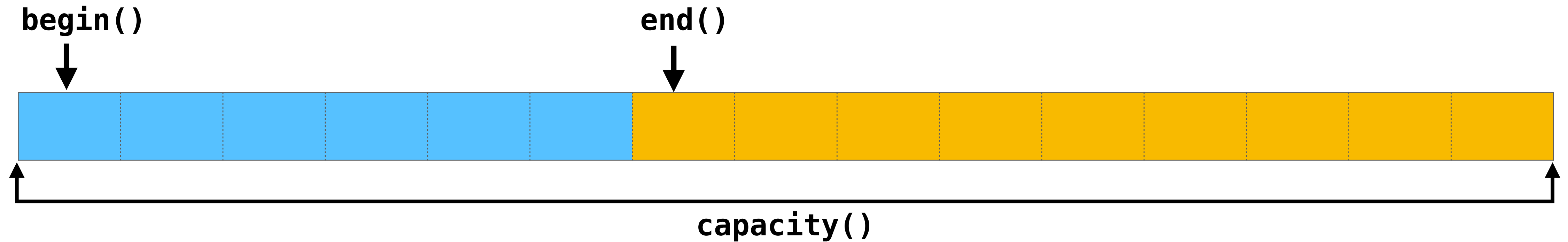
```
void Func()
{
    std::byte my_buffer[12];
    int my_integer = 5;
    ...
    ...
    ...
    ...
    my_buffer[12] = 0;
}
```

# Stack Red Zones



# AddressSanitizer ContainerOverflow

`std::vector<T>`



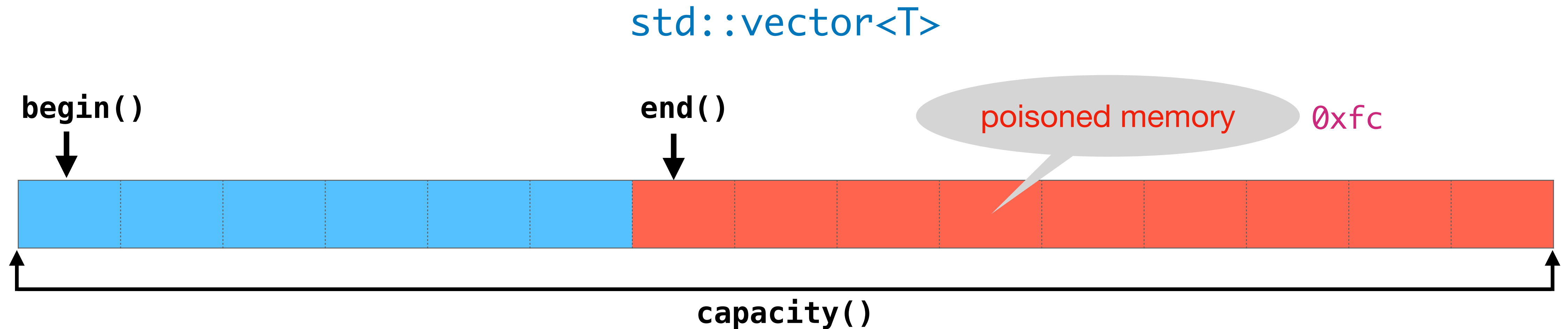
with the help of **code annotations** in `std::vector`

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](https://github.com/google/sanitizers/wiki/AddressSanitizerPerformanceNumbers)

# Problems & Gotchas

Stuff you need to know

**VS 16.7-16.9**

# Compiling/Linking from command-line

Manual CLI compile/link can be tedious  
(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/](https://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.



**/fsanitize:address**

fixed in **v16.9**

# `/ZI`

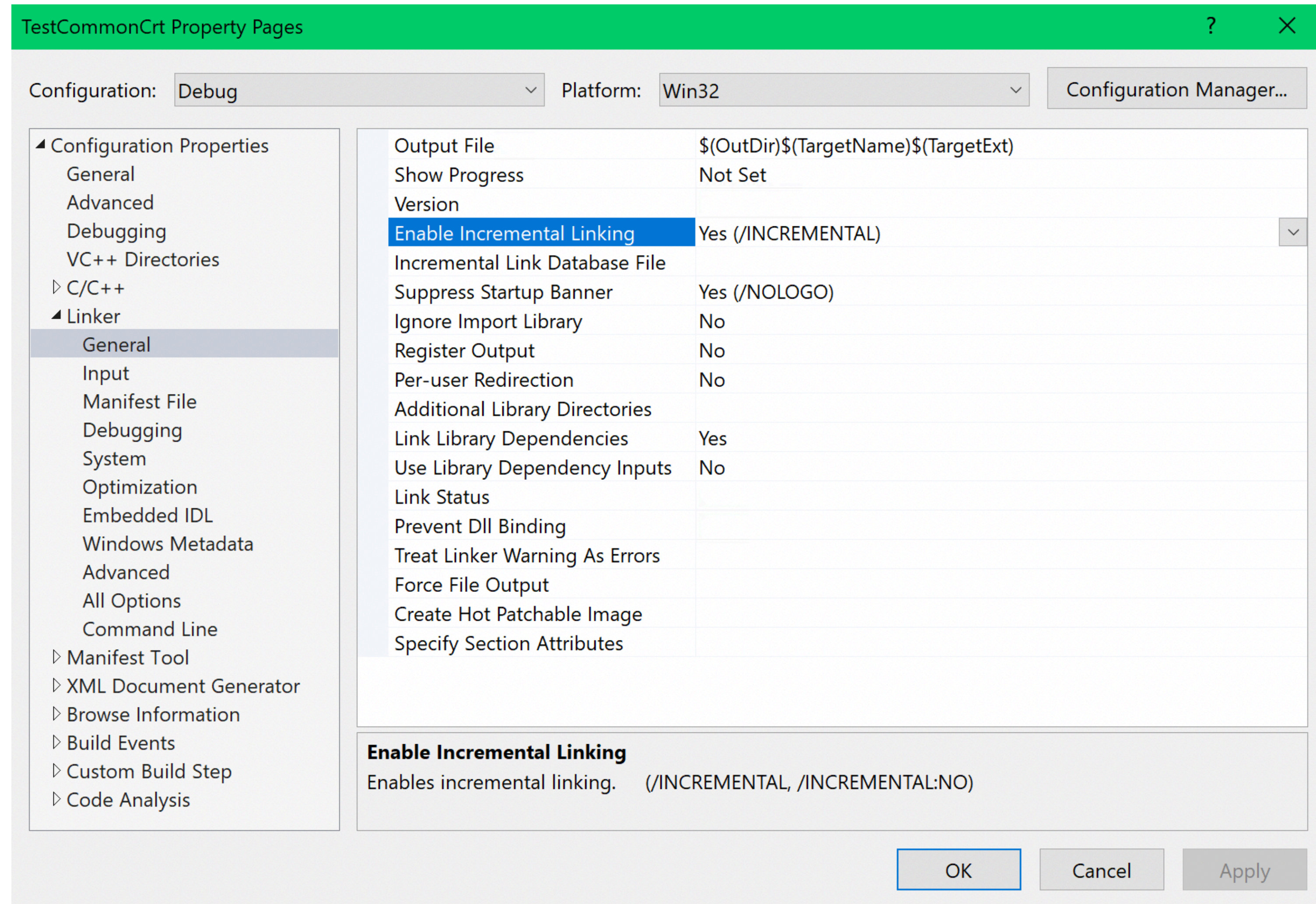
## Edit and Continue (Debug)

error MSB8059:

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

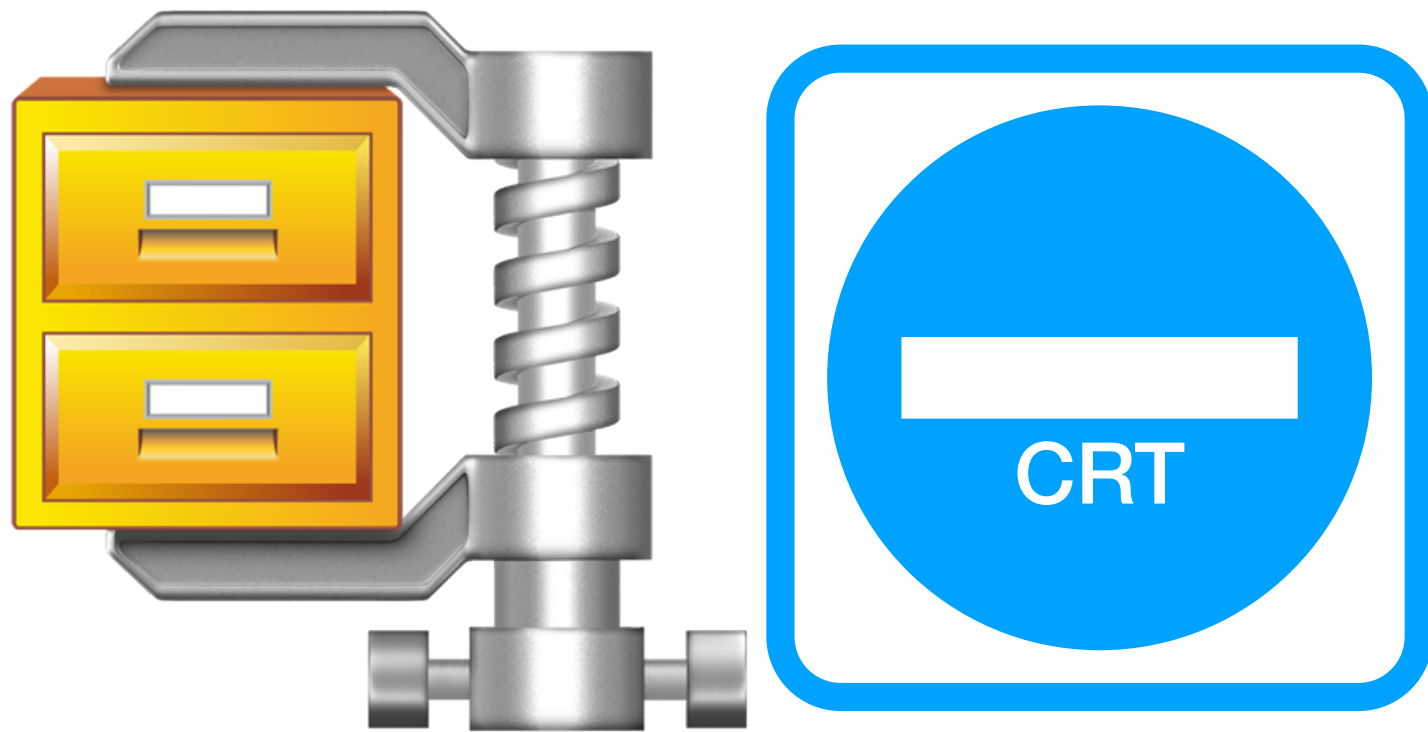
# Link /INCREMENTAL

Debug builds



error MSB8059:

-fsanitize=address (Enable Address Sanitizer) is incompatible with option 'incremental linking (/INCREMENTAL)'



# ASan + /NODEFAULTLIB

The linker will be very mad at you

TestCommonCrt Property Pages

Configuration: All Configurations Platform: Win32 Configuration Manager...

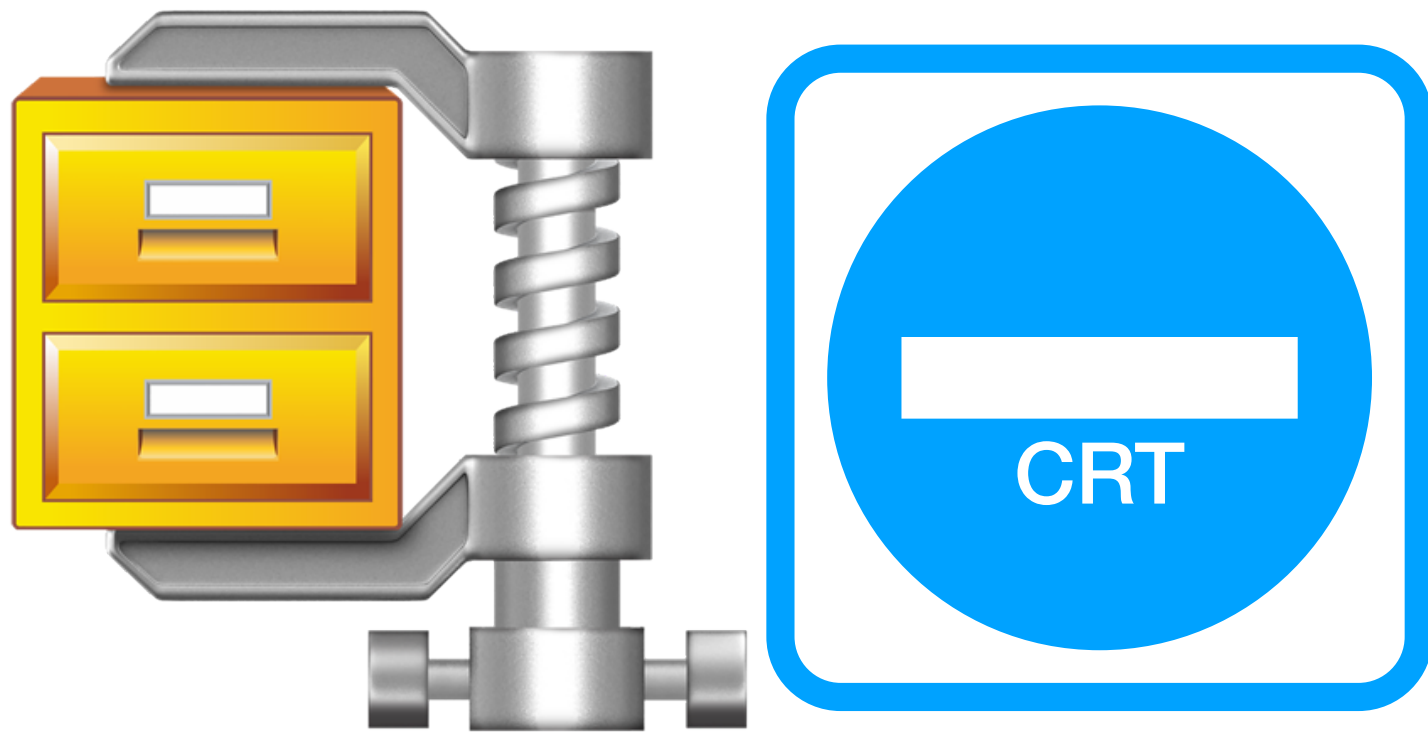
- Configuration Properties
  - General
  - Advanced
  - Debugging
  - VC++ Directories
  - C/C++
  - Linker
    - General
    - Input
    - Manifest File
    - Debugging
    - System
    - Optimization
    - Embedded IDL
    - Windows Metadata
    - Advanced
    - All Options
    - Command Line
  - Manifest Tool
  - XML Document Generator
  - Browse Information
  - Build Events
  - Custom Build Step
  - Code Analysis

Additional Dependencies	msi.lib;%(AdditionalDependencies)
Ignore All Default Libraries	Yes (/NODEFAULTLIB)
Ignore Specific Default Libraries	
Module Definition File	
Add Module to Assembly	
Embed Managed Resource File	
Force Symbol References	
Delay Loaded DLLs	
Assembly Link Resource	

**Ignore All Default Libraries**  
The /NODEFAULTLIB option tells the linker to remove one or more default libraries from the list of libraries it searches when resolving external references.

OK Cancel Apply





# ASan + /NODEFAULTLIB

The linker will be very mad at you  
if you have a custom entry point  
(bypass CRT main)

TestCommonCrt Property Pages

Configuration: All Configurations Platform: Win32 Configuration Manager...

- Configuration Properties
  - General
  - Advanced
  - Debugging
  - VC++ Directories
  - C/C++
  - Linker
    - General
    - Input
    - Manifest File
    - Debugging
    - System
    - Optimization
    - Embedded IDL
    - Windows Metadata
    - Advanced
    - All Options
    - Command Line
  - Manifest Tool
  - XML Document Generator
  - Browse Information
  - Build Events
  - Custom Build Step
  - Code Analysis

Entry Point	
No Entry Point	No
Set Checksum	No
Base Address	
Randomized Base Address	Yes (/DYNAMICBASE)
Fixed Base Address	
Data Execution Prevention (DEP)	Yes (/NXCOMPAT)
Turn Off Assembly Generation	No
Unload delay loaded DLL	
Nobind delay loaded DLL	
Import Library	
Merge Sections	
Target Machine	MachineX86 (/MACHINE:X86)
Profile	No
CLR Thread Attribute	
CLR Image Type	Default image type
Key File	
Key Container	
Delay Sign	
CLR Unmanaged Code Check	

**Entry Point**  
The /ENTRY option specifies an entry point function as the starting address for an .exe file or DLL.

OK Cancel Apply



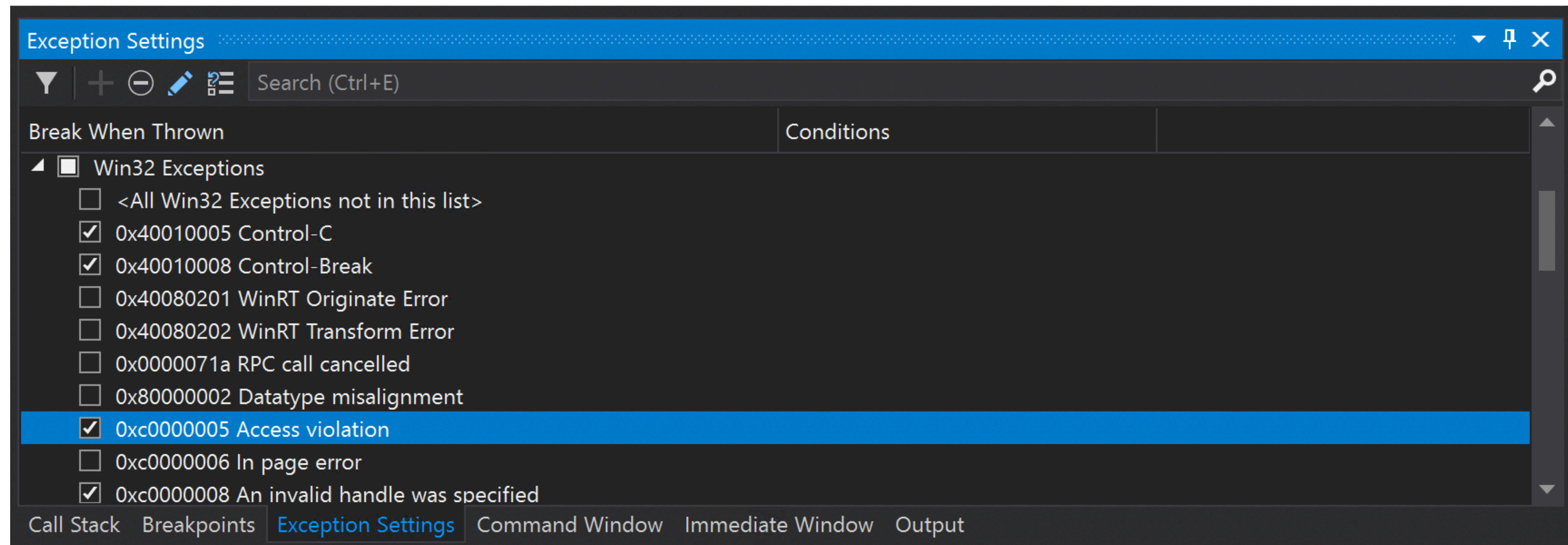
# Access Violation Exceptions

*Debugger* may break frequently and you may see a lot of SEH **access violation** exceptions

This is normal (x64). It's how ASAN traps memory allocations to instrument its own *shadow memory*

Just tell the *Debugger* to stop breaking on this type of exception:

uncheck 



# 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/](https://devblogs.microsoft.com/cppblog/asan-for-windows-x64-and-debug-build-support/)

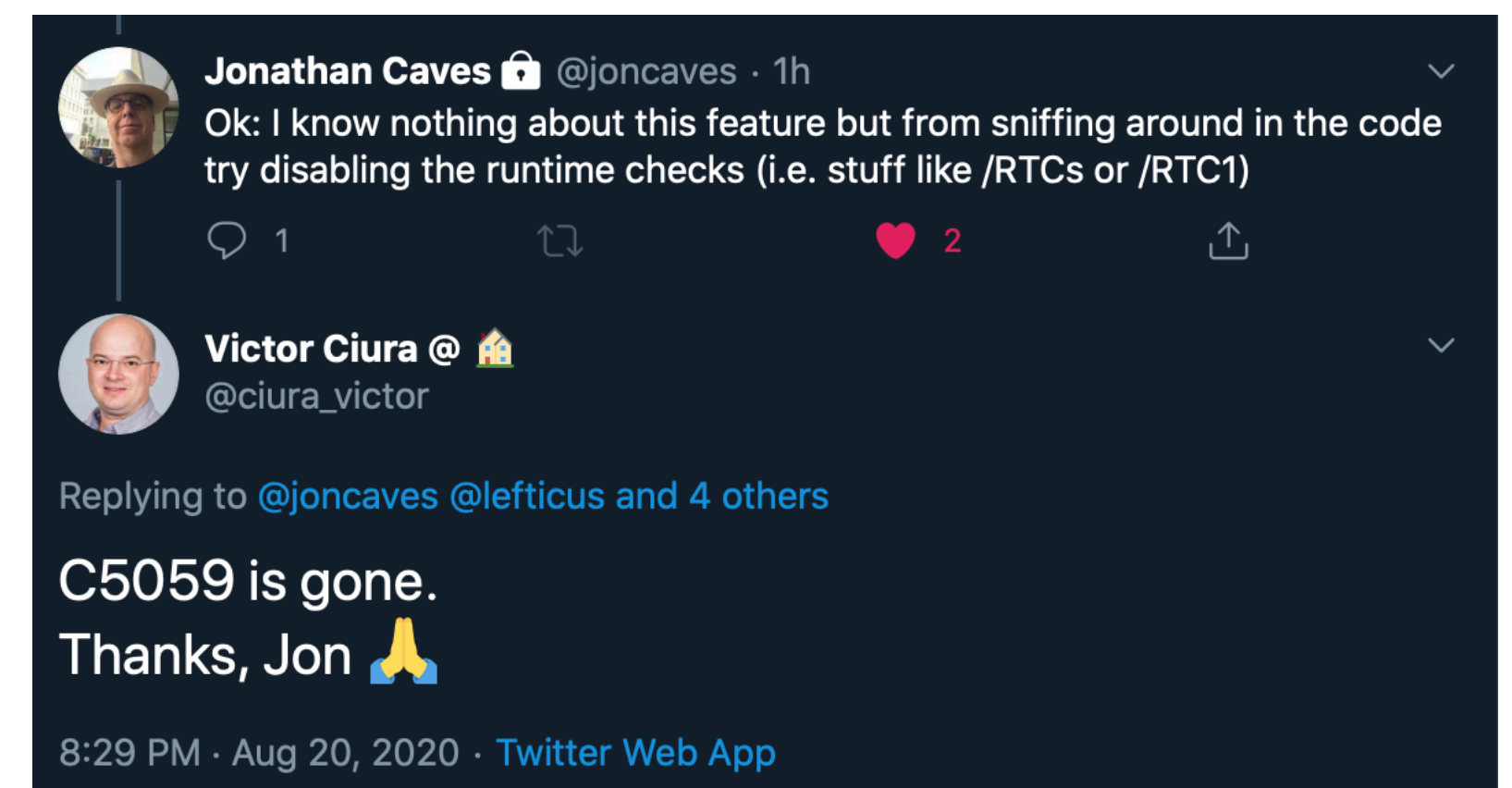
# /RTCs and /RTC1 Runtime Checks

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](https://twitter.com/ciura_victor/status/1296499633825492992)

# Missing PDBs from VS

**v16.7**

**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**

fixed in **v16.9**

# Missing PDBs from VS

**v16.7**

**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**

fixed in **v16.9**

# 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

IDE integration for ASan-reported **exceptions** now handles the complete collection of reportable ASan exceptions

# Linker Trouble?

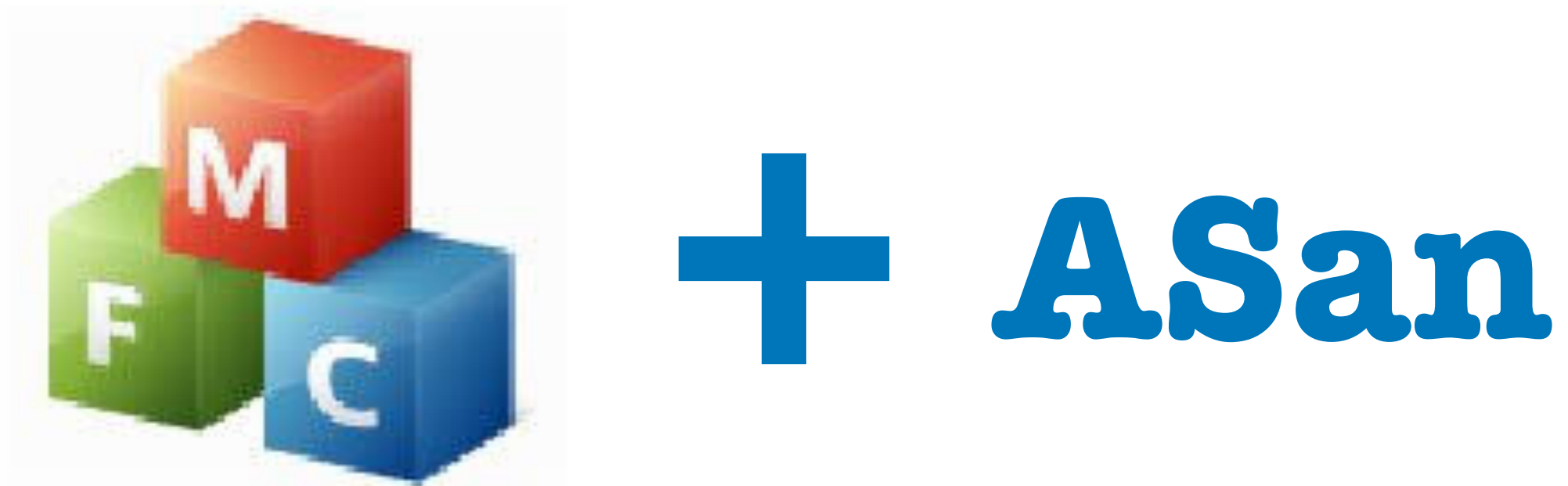
## Building a static LIB, linked into an EXE

### [Debug | x64]

```
>libucrtd.lib(debug_heap.obj) : warning LNK4006: _calloc_dbg already defined in clang_rt.asan_dbg-x86_64.lib(asan_malloc_win.cc.obj); second definition ignored
>libucrtd.lib(debug_heap.obj) : warning LNK4006: _expand_dbg already defined in clang_rt.asan_dbg-x86_64.lib(asan_malloc_win.cc.obj); second definition ignored
>libucrtd.lib(debug_heap.obj) : warning LNK4006: _free_dbg already defined in clang_rt.asan_dbg-x86_64.lib(asan_malloc_win.cc.obj); second definition ignored
>libucrtd.lib(debug_heap.obj) : warning LNK4006: _malloc_dbg already defined in clang_rt.asan_dbg-x86_64.lib(asan_malloc_win.cc.obj); second definition ignored
>libucrtd.lib(debug_heap.obj) : warning LNK4006: _realloc_dbg already defined in clang_rt.asan_dbg-x86_64.lib(asan_malloc_win.cc.obj); second definition ignored
>libucrtd.lib(debug_heap.obj) : warning LNK4006: _realloc_dbg already defined in clang_rt.asan_dbg-x86_64.lib(asan_malloc_win.cc.obj); second definition ignored
>libucrtd.lib(debug_heap.obj) : warning LNK4006: _realloc_dbg already defined in clang_rt.asan_dbg-x86_64.lib(asan_malloc_win.cc.obj); second definition ignored
>libucrtd.lib(debug_heap.obj) : warning LNK4006: _realloc_dbg already defined in clang_rt.asan_dbg-x86_64.lib(asan_malloc_win.cc.obj); second definition ignored
>libucrtd.lib(expand.obj) : warning LNK4006: _expand already defined in clang_rt.asan_dbg-x86_64.lib(asan_malloc_win.cc.obj); second definition ignored
```

### [Debug | x86]

```
>libucrtd.lib(debug_heap.obj) : warning LNK4006: __calloc_dbg already defined in clang_rt.asan_dbg-i386.lib(asan_malloc_win.cc.obj); second definition ignored
>libucrtd.lib(debug_heap.obj) : warning LNK4006: __expand_dbg already defined in clang_rt.asan_dbg-i386.lib(asan_malloc_win.cc.obj); second definition ignored
>libucrtd.lib(debug_heap.obj) : warning LNK4006: __free_dbg already defined in clang_rt.asan_dbg-i386.lib(asan_malloc_win.cc.obj); second definition ignored
>libucrtd.lib(debug_heap.obj) : warning LNK4006: __malloc_dbg already defined in clang_rt.asan_dbg-i386.lib(asan_malloc_win.cc.obj); second definition ignored
>libucrtd.lib(debug_heap.obj) : warning LNK4006: __realloc_dbg already defined in clang_rt.asan_dbg-i386.lib(asan_malloc_win.cc.obj); second definition ignored
>libucrtd.lib(debug_heap.obj) : warning LNK4006: __realloc_dbg already defined in clang_rt.asan_dbg-i386.lib(asan_malloc_win.cc.obj); second definition ignored
>libucrtd.lib(debug_heap.obj) : warning LNK4006: __realloc_dbg already defined in clang_rt.asan_dbg-i386.lib(asan_malloc_win.cc.obj); second definition ignored
>libucrtd.lib(debug_heap.obj) : warning LNK4006: __realloc_dbg already defined in clang_rt.asan_dbg-i386.lib(asan_malloc_win.cc.obj); second definition ignored
>libucrtd.lib(expand.obj) : warning LNK4006: __expand already defined in clang_rt.asan_dbg-i386.lib(asan_malloc_win.cc.obj); second definition ignored
```



```
>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](https://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

**ASAN Finds bugs**

**Really !**

AddressSanitizer: **heap-buffer-overflow** on address 0x0a2301b4 pc 0x005b7a35 bp 0x011df078 sp 0x011df06c  
READ of size 5 at 0x0a2301b4 thread T0

```
#0 0x5b7a4d in __asan_wrap_strlen crt\asan\llvm\compiler-rt\lib\sanitizer_common\sanitizer_common_interceptors.inc:365
#1 0x278eeb in ATL::CStringT<char,0>::StringLength MSVC\14.28.29333\atlmfc\include\atlsimpstr.h:726
#2 0x278a35 in ATL::CStringT<char,0>::SetString MSVC\14.28.29333\atlmfc\include\atlsimpstr.h:602
#3 0x274d69 in ATL::CStringT<char,0>::operator= MSVC\14.28.29333\atlmfc\include\atlsimpstr.h:314
#4 0x274d99 in ATL::CStringT<char,ATL::StrTraitATL<char,ATL::ChTraitsCRT<char>>>::operator=
    MSVC\14.28.29333\atlmfc\include\cstringt.h:1315
#5 0x27469c in ATL::CStringT<char,ATL::StrTraitATL<char,ATL::ChTraitsCRT<char>>>::CStringT
    MSVC\14.28.29333\atlmfc\include\cstringt.h:1115
#6 0x27641a in SerValUtil::DecryptString C:\JobAI\advinst\msicomp\serval\SerValUtil.cpp:85
#7 0x3e1660 in TestSerVal C:\JobAI\testunits\serval\SerValTests.cpp:60
#8 0x5880e5 in FunctionTest::Run C:\JobAI\testunits\Tester.cpp:71
#9 0x5889b1 in Tester::RunTest C:\JobAI\testunits\Tester.cpp:186
#10 0x586ddb in Tester::ExecuteCommandLine C:\JobAI\testunits\Tester.cpp:558
#11 0x5798d1 in main C:\JobAI\testunits\comps\TestComponents.cpp:2236
```

0x0a2301b4 is located 0 bytes to the right of 4-byte region [0x0a2301b0,0x0a2301b4)  
allocated by thread T0

# Fun with ATL::CString

```
ATL::CSimpleArray<BYTE> decrypted;  
X::DecryptString(encrypted, decrypted);  
  
ATL::CStringA decryptedStr(&decrypted[0]);  
decryptedStr.ReleaseBufferSetLength(decrypted.GetSize());
```

# Fun with ATL::CString

```
ATL::CSimpleArray<BYTE> decrypted;  
X::DecryptString(encrypted, decrypted);  
  
ATL::CStringA decryptedStr(&decrypted[0]);  
decryptedStr.ReleaseBufferSetLength(decrypted.GetSize());
```

# Fun with ATL::CString

Somewhere inside

```
ATL::CString::ReleaseBufferSetLength(int nLength)
{
    GetData()->nDataLength = nLength;
    m_pszData[nLength] = 0;
    ...
}
```

# Fun with `ATL::CString`

Classic story: null-terminated string.

`Array` of chars to `string` class - `size` has a different meaning, because of the ending `\0`

# Easy fix

```
ATL::CSimpleArray<BYTE> decrypted;  
X::DecryptString(encrypted, decrypted);  
  
ATL::CStringA decryptedStr(decrypted.GetData(), decrypted.GetSize());
```

It's actually more efficient, too.



AddressSanitizer: **stack-buffer-overflow** on address 0x00b3f766 at pc 0x00181b07 bp 0x00b3f6bc sp 0x00b3f6b0

WRITE of size 2 at 0x00b3f766 thread T0

```
#0 0x181b06 in CommonCrt::ItoaT<wchar_t> C:\JobAI\platform\util\CommonCrt.h:402
#1 0x183e02 in CommonCrt::Itoa C:\JobAI\platform\util\CommonCrt.cpp:119
#2 0x190696 in TestCommonCrtItoa C:\JobAI\testunits\common_crt\CommonCrtTests.cpp:93
#3 0x194821 in Tester::RunTest<int (__cdecl*)(void)> C:\JobAI\testunits\common_crt\tester\Tester.h:55
#4 0x194b65 in main C:\JobAI\testunits\common_crt\main.cpp:22
#5 0x1cc142 in invoke_main crt\vcstartup\src\startup\exe_common.inl:78
#6 0x1cc046 in __scrt_common_main_seh crt\vcstartup\src\startup\exe_common.inl:288
#7 0x1cbeec in __scrt_common_main crt\vcstartup\src\startup\exe_common.inl:330
#8 0x1cc1a7 in mainCRTStartup crt\vcstartup\src\startup\exe_main.cpp:16
#9 0x7645fa28 in BaseThreadInitThunk+0x18 (C:\WINDOWS\System32\KERNEL32.DLL+0x6b81fa28)
#10 0x773e76b3 in RtlGetAppContainerNamedObjectPath+0xe3 (C:\WINDOWS\SYSTEM32\ntdll.dll+0x4b2e76b3)
#11 0x773e7683 in RtlGetAppContainerNamedObjectPath+0xb3 (C:\WINDOWS\SYSTEM32\ntdll.dll+0x4b2e7683)
```

Address 0x00b3f766 is located in stack of thread T0 at offset 30 in frame

```
#0 0x1905ef in TestCommonCrtItoa C:\JobAI\testunits\common_crt\CommonCrtTests.cpp:84
```

This frame has 2 object(s):

```
[16, 30) 'result1' <== Memory access at offset 30 overflows this variable
[32, 46) 'result2' <== Memory access at offset 30 underflows this variable
```

# Naive Test Unit

```
const LONG      kNumber1 = 21474835;  
TCHAR          result1[kMaxSize];  
const TCHAR *  compare1 = L"21474835";  
const LONG      kNumber2 = -2100;  
TCHAR          result2[kMaxSize];  
const TCHAR *  compare2 = L"-2100";  
  
CommonCrt::Itoa(kNumber1, result1);  
  
ASSERT_EQ(CompareStrings(result1, compare1));  
...
```

# Naive Test Unit

```
const LONG      kNumber1 = 21474835;  
TCHAR          result1[kMaxSize];  
const TCHAR *  compare1 = L"21474835";  
const LONG      kNumber2 = -2100;  
TCHAR          result2[kMaxSize];  
const TCHAR *  compare2 = L"-2100";  
  
CommonCrt::Itoa(kNumber1, result1);  
  
ASSERT_EQ(CompareStrings(result1, compare1));  
...
```

AddressSanitizer: **stack-buffer-overflow** on address 0x00843b3ae544 at pc 0x7ff6da711d86 bp 0x00843b3ae180  
sp 0x00843b3ae188  
READ of size 1 at 0x00843b3ae544 thread T0

```
#0 0x7ff6da711d85 in std::_Char_traits<unsigned char, long>::length MSVC\14.28.29333\include\xstring:143
#1 0x7ff6da711667 in std::basic_string<unsigned char, std::char_traits<unsigned char>, std::allocator<unsigned char> >::assign
    MSVC\14.28.29333\include\xstring:3062
#2 0x7ff6da70af94 in std::basic_string<unsigned char...> MSVC\14.28.29333\include\xstring:2417
#3 0x7ff6da70c163 in TestStringUtilAsciiToUnicode C:\JobAI\testunits\strings\StringEncodingTests.cpp:26
#4 0x7ff6da98db80 in FunctionTest::Run C:\JobAI\testunits\Tester.cpp:71
#5 0x7ff6da98fb05 in Tester::RunTest C:\JobAI\testunits\Tester.cpp:186
#6 0x7ff6da98b3b4 in Tester::ExecuteCommandLine C:\JobAI\testunits\Tester.cpp:558
#7 0x7ff6da97b59e in main C:\JobAI\testunits\comps\TestComponents.cpp:2236
#8 0x7ff6dac2a8d8 in invoke_main d:\agent\_work\63\s\src\vctools\crt\vcstartup\src\startup\exe_common.inl:78
```

Address 0x00843b3ae544 is located in stack of thread T0 at offset 564 in frame  
#0 0x7ff6da70badf in TestStringUtilAsciiToUnicode C:\JobAI\testunits\strings\StringEncodingTests.cpp:14

This frame has 12 object(s):


```
[32, 72) 'result1'
[48, 88) 'kTextString1'
[64, 104) 'result2'
[80, 120) 'kTextString3'
[96, 136) 'result3'
[112, 152) 'compiler temporary'
[128, 144) 'compiler temporary'
[144, 160) 'compiler temporary'
[160, 164) 'uChars'
[176, 177) 'compiler temporary'
[192, 216) 'compiler temporary'
[208, 232) 'compiler temporary' <== Memory access at offset 564 overflows this variable
```

# Naive Test Unit

```
unsigned char          uChars[] = { 0x41, 0x42, 0x43, 0x44 };
const basic_string<unsigned char> kTextString3(uChars);
wstring              result3 = wstring(kTextString3.begin(), kTextString3.end());
if (StringUtil::AsciiToUnicode(kTextString3)  $\neq$  result3)
    return -1;
```

# Naive Test Unit

```
unsigned char          uChars[] = { 0x41, 0x42, 0x43, 0x44 };
const basic_string<unsigned char> kTextString3(uChars);
wstring               result;
if (StringUtil::AsciiToUnicode(kTextString3, result))
    return -1;
return 0;
```

 (local variable) const `std::basic_string`<unsigned char> kTextString3

[Search Online](#)

C6054: String 'uChars' might not be zero-terminated.

# Naive Test Unit

```
unsigned char          uChars[] = { 0x41, 0x42, 0x43, 0x44 };
const basic_string<unsigned char> kTextString3(uChars);
wstring                result;
if (StringUtil::AsciiToUnicode(kTextString3, result))
    return -1;
return 0;
```

(local variable) const `std::basic_string`<unsigned char> kTextString3  
Search Online  
C6054: String 'uChars' might not be zero-terminated.

It's worth paying attention to your squiggles !  
VS analyzer does a pretty good job keeping you safe.

AddressSanitizer: **global-buffer-overflow** on address 0x00c158ca at pc 0x00838b91 bp 0x016fef98 sp 0x016fef8c

READ of size 2 at 0x00c158ca thread T0

```
#0 0x838b90 in StringUtil::StoreNULLSeparatedStrings C:\JobAI\platform\util\strings\StringProcessing.cpp:430
#1 0x67edfb in TestStringUtilStoreNULLSeparatedStrings C:\JobAI\testunits\strings\StringProcessingTests.cpp:563
#2 0x7e8035 in FunctionTest::Run C:\JobAI\testunits\Tester.cpp:71
#3 0x7e8901 in Tester::RunTest C:\JobAI\testunits\Tester.cpp:186
#4 0x7e6d2b in Tester::ExecuteCommandLine C:\JobAI\testunits\Tester.cpp:558
#5 0x7d9821 in main C:\JobAI\testunits\comps\TestComponents.cpp:2236
#6 0x9d92f2 in invoke_main crt\vcstartup\src\startup\exe_common.inl:78
#7 0x9d91f6 in __scrt_common_main_seh crt\vcstartup\src\startup\exe_common.inl:288
#8 0x9d909c in __scrt_common_main crt\vcstartup\src\startup\exe_common.inl:330
#9 0x9d9357 in mainCRTStartup crt\vcstartup\src\startup\exe_main.cpp:16
```

0x00c158ca is located 0 bytes to the right of global variable '**<C++ string literal>**' defined in 'StringProcessingTests.cpp:561:9' (0xc158a0) of size 42

#### SUMMARY:

AddressSanitizer: global-buffer-overflow StringProcessing.cpp:430 in StringUtil::StoreNULLSeparatedStrings



# Use the full power of your Debugger

The screenshot displays a debugger interface with two main panels: Autos and Call Stack.

**Autos Window:** This panel shows a search bar with the text "Search (Ctrl+E)" and a search icon. Below it, a table lists variables with their names, values, and types. A search icon and a search depth dropdown set to "3" are also visible.

Name	Value	Type
*st	116 't'	const wchar_t
aBuff	0x00007ff7e82ac100 L"token0"	const wchar_t *
secBuf2	L""	std::wstring
st	0x00007ff7e82ac11c L"token2"	const wchar_t *

**Call Stack Window:** This panel shows a list of function calls. The top entry is highlighted with a yellow arrow.

Name	Language
TestPlatform.exe!StringUtil::StoreNULLSeparatedStrings(const wchar_t * aBuff, std::vector<std::wstring, std::allocator<std::wstring>...)	C++
TestPlatform.exe!TestStringUtilStoreNULLSeparatedStrings() Line 564	C++
TestPlatform.exe!FunctionTest::Run() Line 71	C++
TestPlatform.exe!Tester::RunTest(const char * aTestName, bool aForce, std::wstring * aTestLog) Line 186	C++
TestPlatform.exe!Tester::ExecuteCommandLine(int argc, char ** argv) Line 550	C++
TestPlatform.exe!main(int argc, char ** argv) Line 2237	C++
TestPlatform.exe!invoke_main() Line 79	C++
TestPlatform.exe!_sCRT_common_main_seh() Line 288	C++
TestPlatform.exe!_sCRT_common_main() Line 331	C++
TestPlatform.exe!mainCRTStartup() Line 17	C++

# Use the full power of your Debugger

The screenshot displays a debugger interface with the following details:

- Process:** [24324] TestPlatform.exe
- Thread:** [24232] Main Thread
- Stack Frame:** StringUtil::StoreNULLSeparatedStrings

The **Memory 1** window shows a memory dump starting at address `0x00007FF7E82AC0F4`. The dump consists of hexadecimal values and their corresponding ASCII representations. The visible text includes:

```
.....t.o.k.e.n.0...t.o.k.e.n.1...  
t.o.k.e.n.2...ù.....  
.....  
.....t.e.s.t.1.,.t.e.s.t.  
2.,.t.e.s.t.3...ù.....  
.....
```

The **StringProcessingTests.cpp** window shows the current stack frame:

```
Platform → StringUtil → StoreNULLSeparatedStrings(const wchar_t * aBuff, vector<wstring>& StringList)
```

# Excessive Test Unit

```
...  
  
buff = L"token0\0token1\0token2\0";  
  
list.clear();  
StringUtil::StoreNULLSeparatedStrings(buff, list);  
  
if (list.size() != 3)  
    return -1;  
if (list[2] != L"token2")  
    return -1;  
  
...
```

# Excessive Test Unit

```
...  
  
buff = L"token0\0token1\0token2\0";  
  
list.clear();  
StringUtil::StoreNULLSeparatedStrings(buff, list);  
  
if (list.size() != 3)  
    return -1;  
if (list[2] != L"token2")  
    return -1;  
  
...
```

# Excessive Test Unit

```
/**
 * Creates a vector with strings that are separated by \0
 * aBuff – buffer containing NULL separated strings
 * aLen – the length of buffer
 * aSection – vector that contains the strings from aBuff
 */
void StoreNULLSeparatedStrings(const wchar_t * aBuff, DWORD aLen,
                               vector<wstring> & aStringList);

/**
 * Creates a vector with strings that are separated by \0 and end with \0\0
 * aBuff – buffer containing NULL separated strings
 * aSection – vector that contains the strings from aBuff
 */
void StoreNULLSeparatedStrings(const wchar_t * aBuff, vector<wstring> & aStringList);
```

# Excessive Test Unit

```
/**  
 * Creates a vector with strings that are separated by \0  
 * aBuff – buffer containing NULL separated strings  
 * aLen – the length of buffer  
 * aSection – vector that contains the strings from aBuff  
 */
```

```
void StoreNULLSeparatedStrings(const wchar_t * aBuff, DWORD  
                               vector<wstring> & aString
```

OUT OF CONTRACT CALL

```
/**  
 * Creates a vector with strings that are separated by \0 and end with \0\0  
 * aBuff – buffer containing NULL separated strings  
 * aSection – vector that contains the strings from aBuff  
 */
```

```
void StoreNULLSeparatedStrings(const wchar_t * aBuff, vector<wstring> & aStringList);
```

**Just enough to wet your appetite**

**Go explore on your own...**





# Explore Further

AddressSanitizer (ASan) for Windows with MSVC

[devblogs.microsoft.com/cppblog/addresssanitizer-asan-for-windows-with-msvc/](https://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/](https://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**

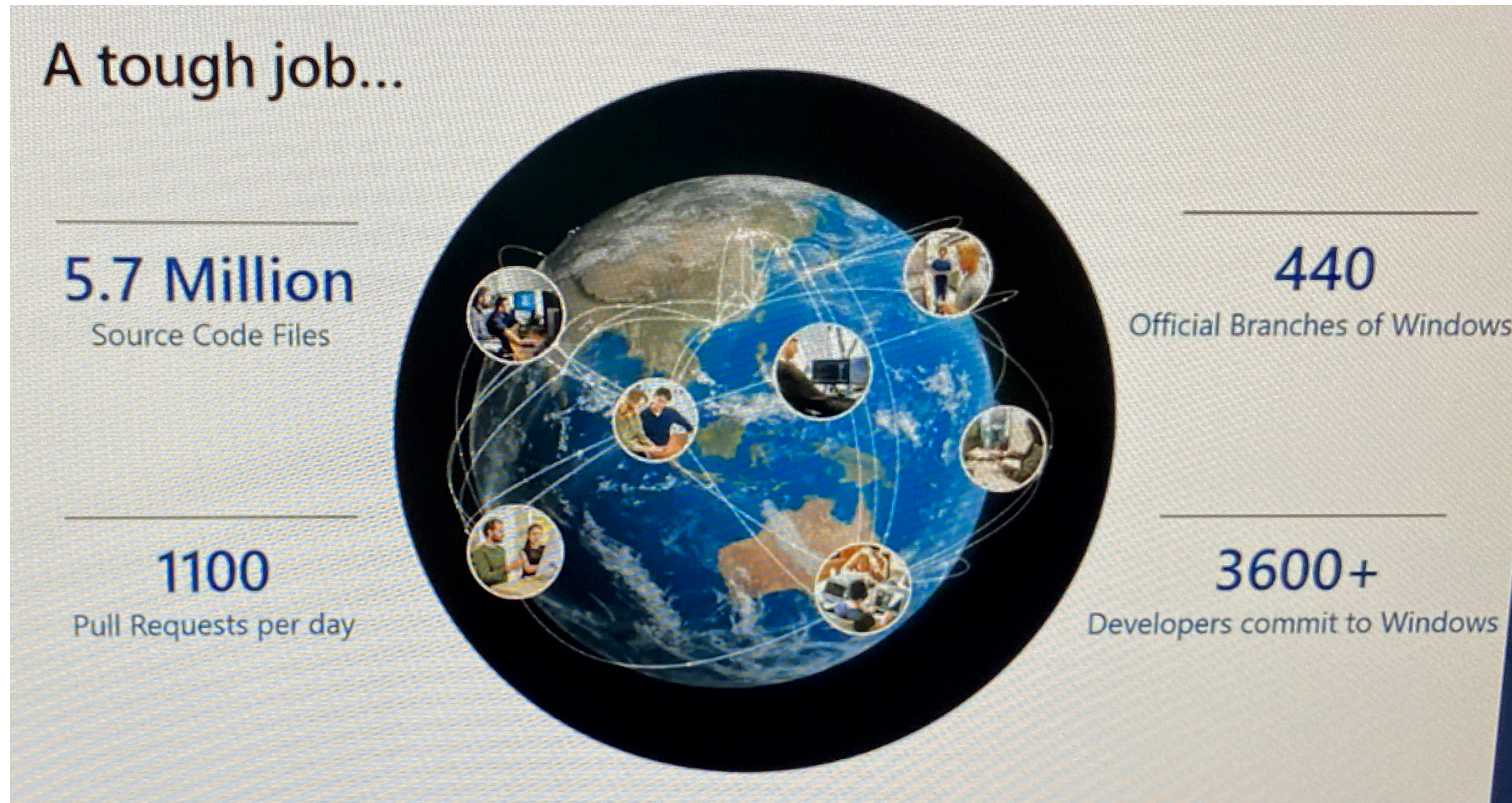
# Sanitizers + Fuzzing



**Automatically generate inputs to you program to crash it.**

# Sanitizers + Fuzzing

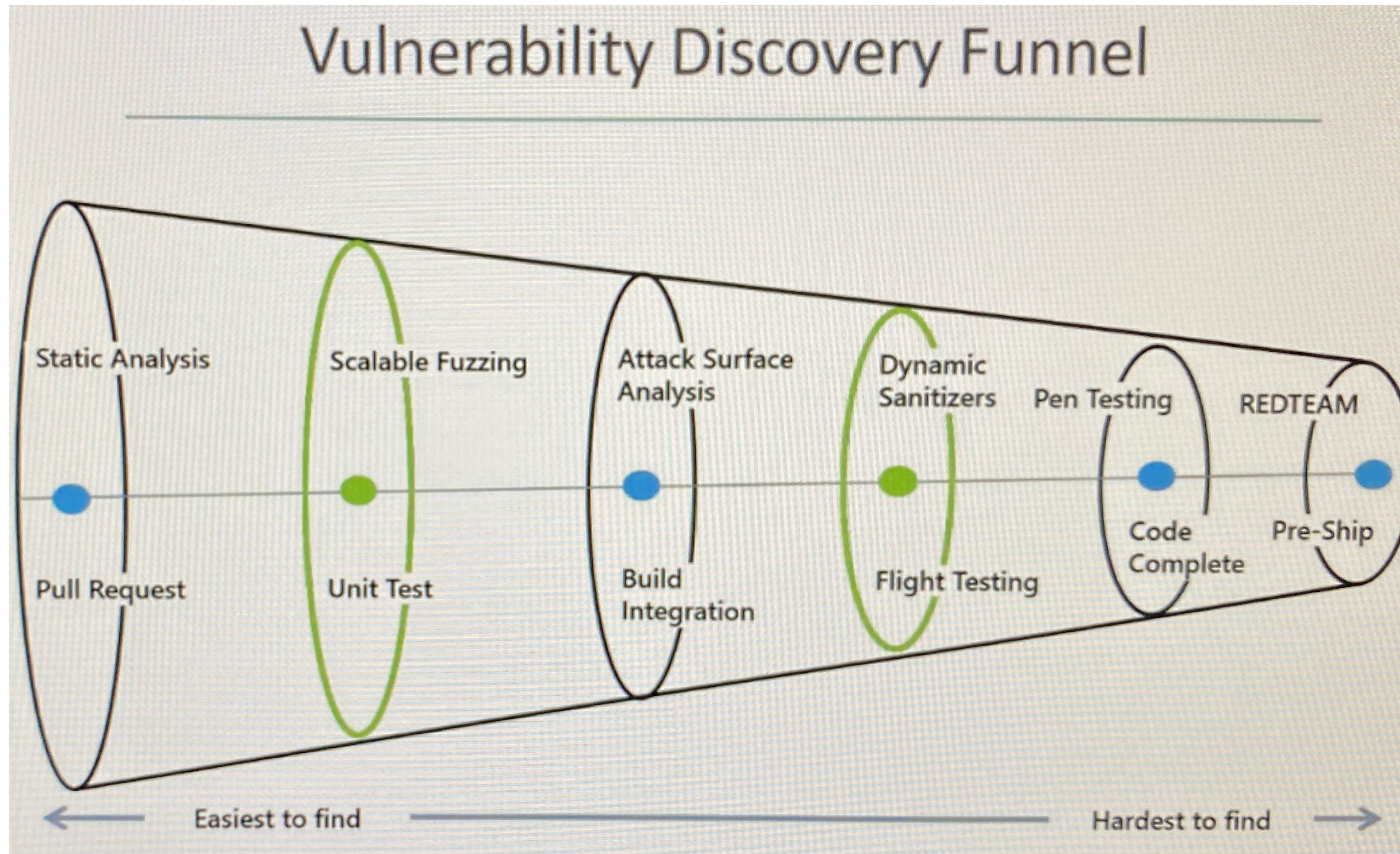
## Case study at Microsoft Windows scale



<https://sched.co/e7C0>

# Sanitizers + Fuzzing

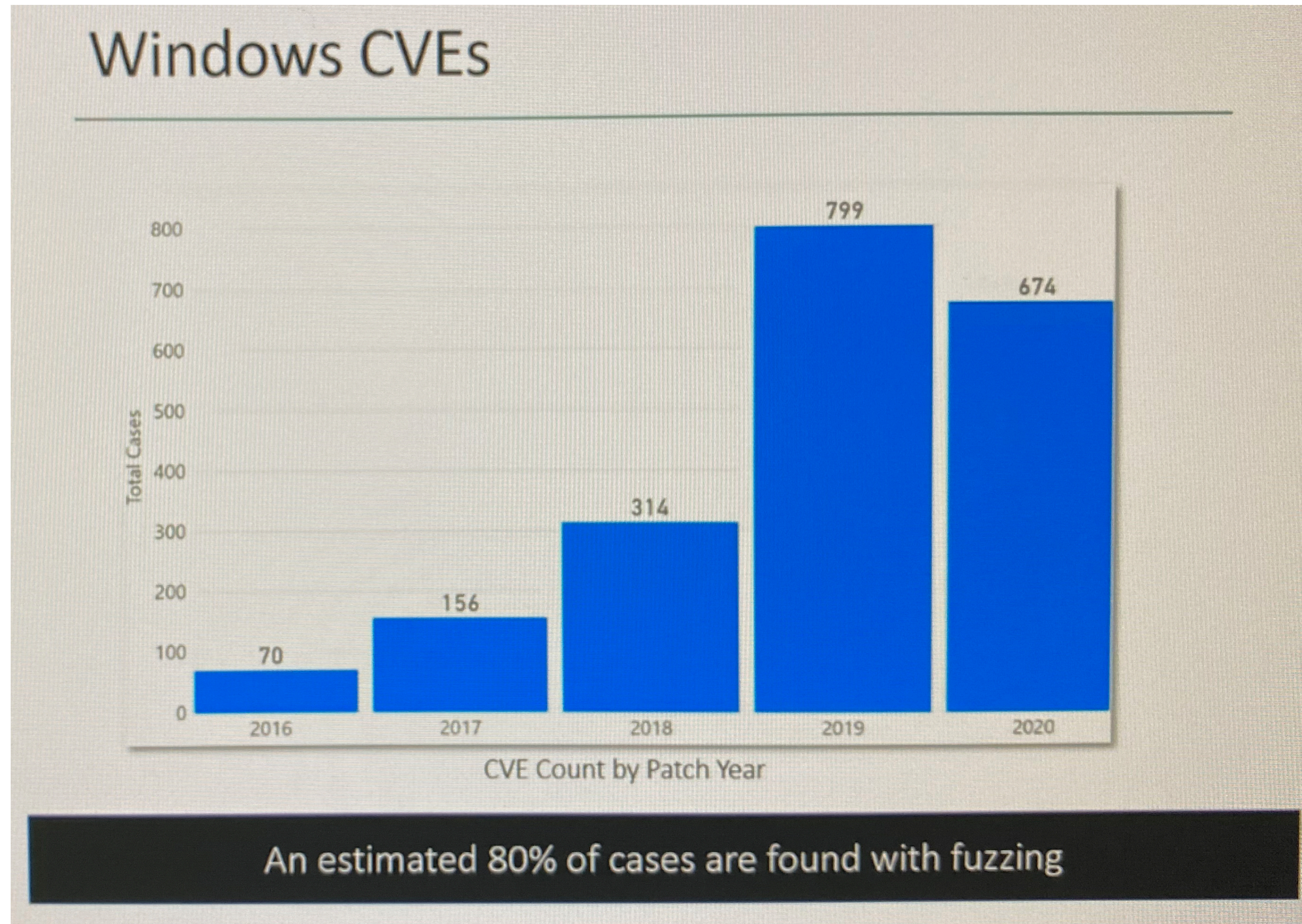
## Case study at Microsoft Windows scale



<https://sched.co/e7C0>

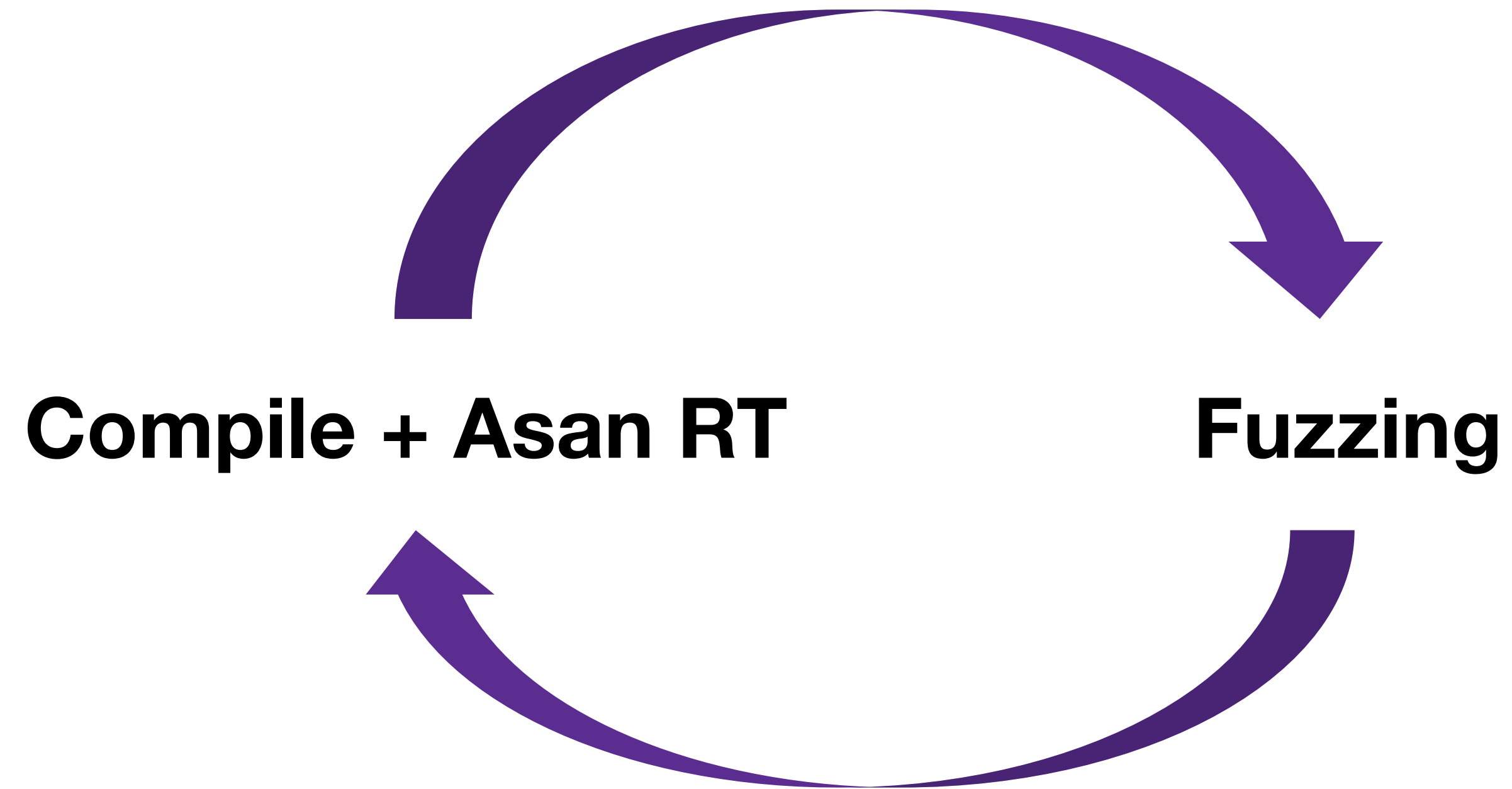
# Sanitizers + Fuzzing

## Case study at Microsoft Windows scale



<https://sched.co/e7C0>

# Workflow





# { ASan + Fuzzing } => Azure

## What is Microsoft Security Risk Detection?

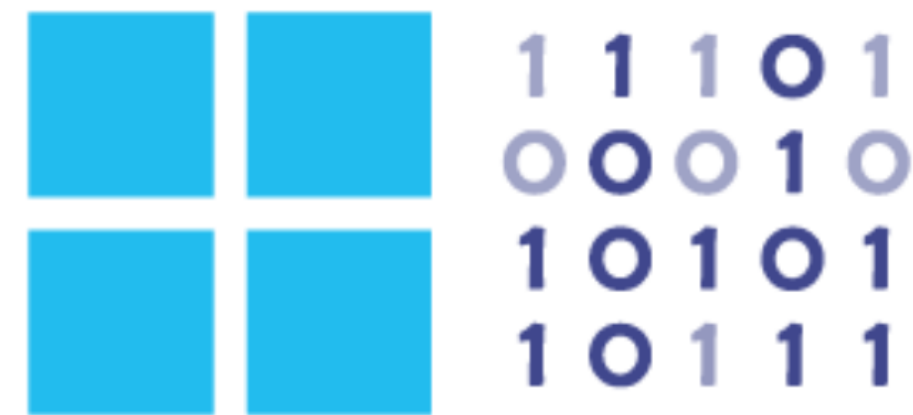
Security Risk Detection is Microsoft's unique fuzz testing service for finding security critical bugs in software. Security Risk Detection helps customers quickly adopt practices and technology battle-tested over the last 15 years at Microsoft.

[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.



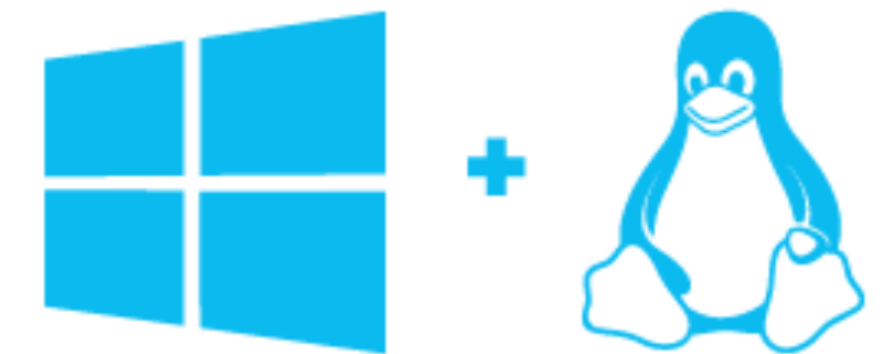
### Battle tested tech

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.

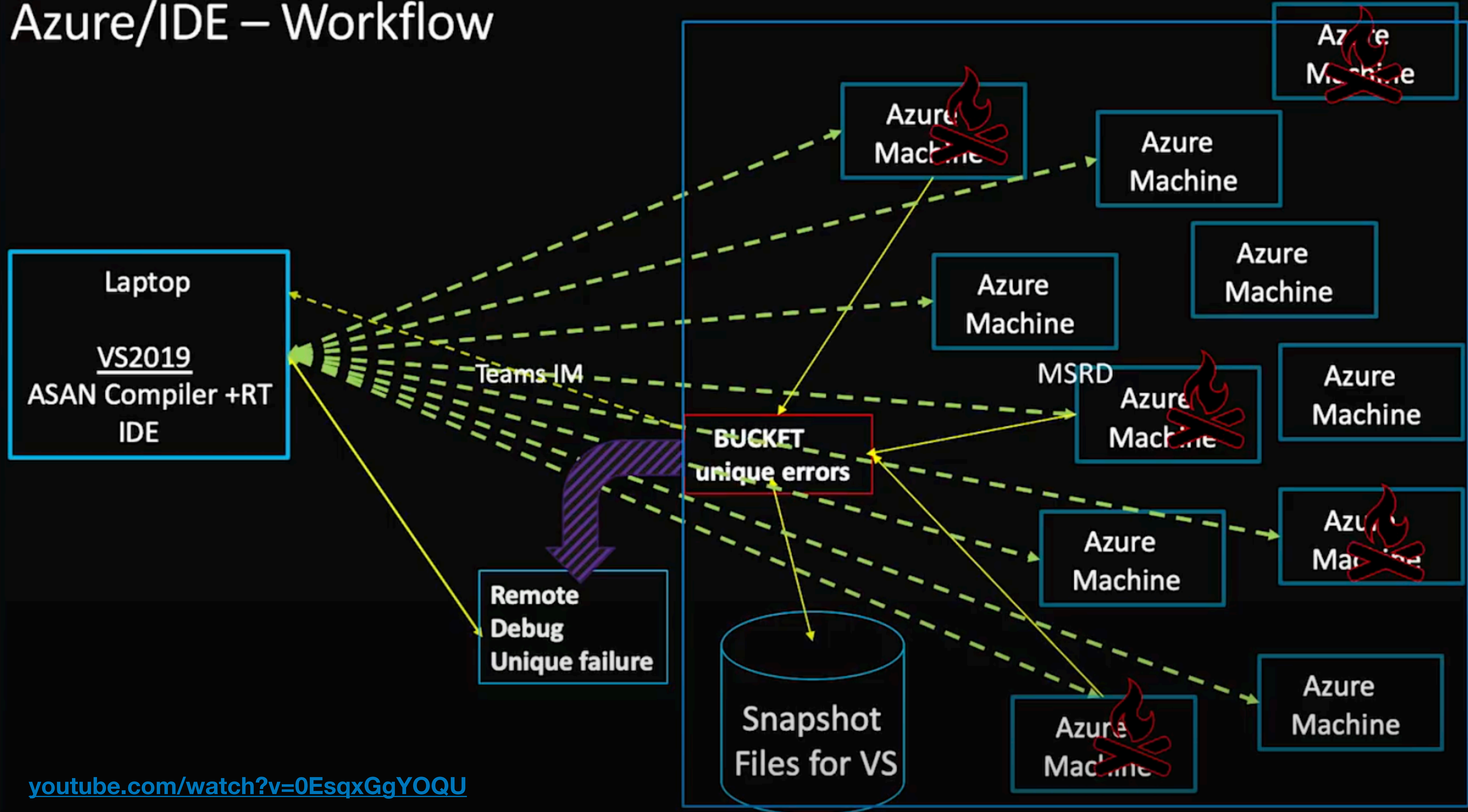


### Cross-platform support

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](https://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

print("please select exactly

-- OPERATOR CLASSES --

types.Operator):
X mirror to the selected
object.mirror_mirror_x"
mirror 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

# 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/](https://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](https://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](https://github.com/microsoft/onefuzz-samples)

# { ASan + Fuzzing } => Azure

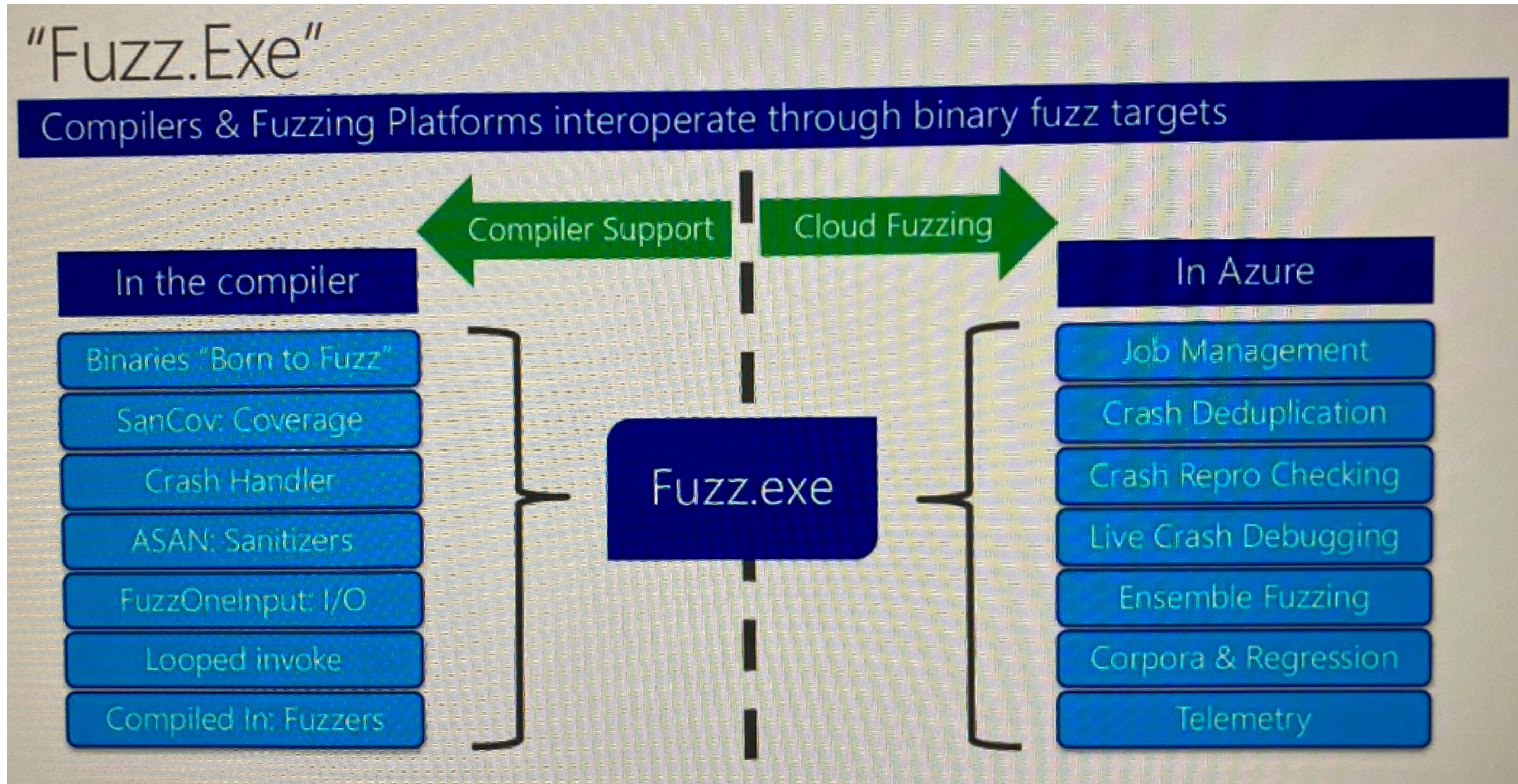
The screenshot shows the Azure Security Risk Detection console. The browser address bar displays the URL: <https://sf-web-ignite.fe-ignite.p.azurewebsites.net/accounts/37fc5d3a-b4b7-4b25-82...>. The user is logged in as Jim Radigan. The navigation menu includes Security Risk Detection, Fuzzing Jobs (selected), Web Scanning, and Learn More. The main heading is "Fuzzing Jobs" with a "Create Job" button. Below is a table of active and completed fuzzing jobs.

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: View, Delete, Stop, Refresh]
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: View, Delete, Stop, Refresh]
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: View, Delete, Stop, Refresh]
6852ebcc	Package CppConFuzzTargetVcAsan	Windows Server 2019 Datacenter x64	9/13/19 9:11 AM	Stopped	5	[Icons: View, Delete, Stop, Refresh]
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: View, Delete, Stop, Refresh]
a3d2b069	Package CppConFuzzTargetVcAsan	Windows Server 2019 Datacenter x64	9/7/19 11:46 PM	Stopped	5	[Icons: View, Delete, Stop, Refresh]

**Azure MSRD service**

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

# { ASan + Fuzzing }



<https://sched.co/e7C0>

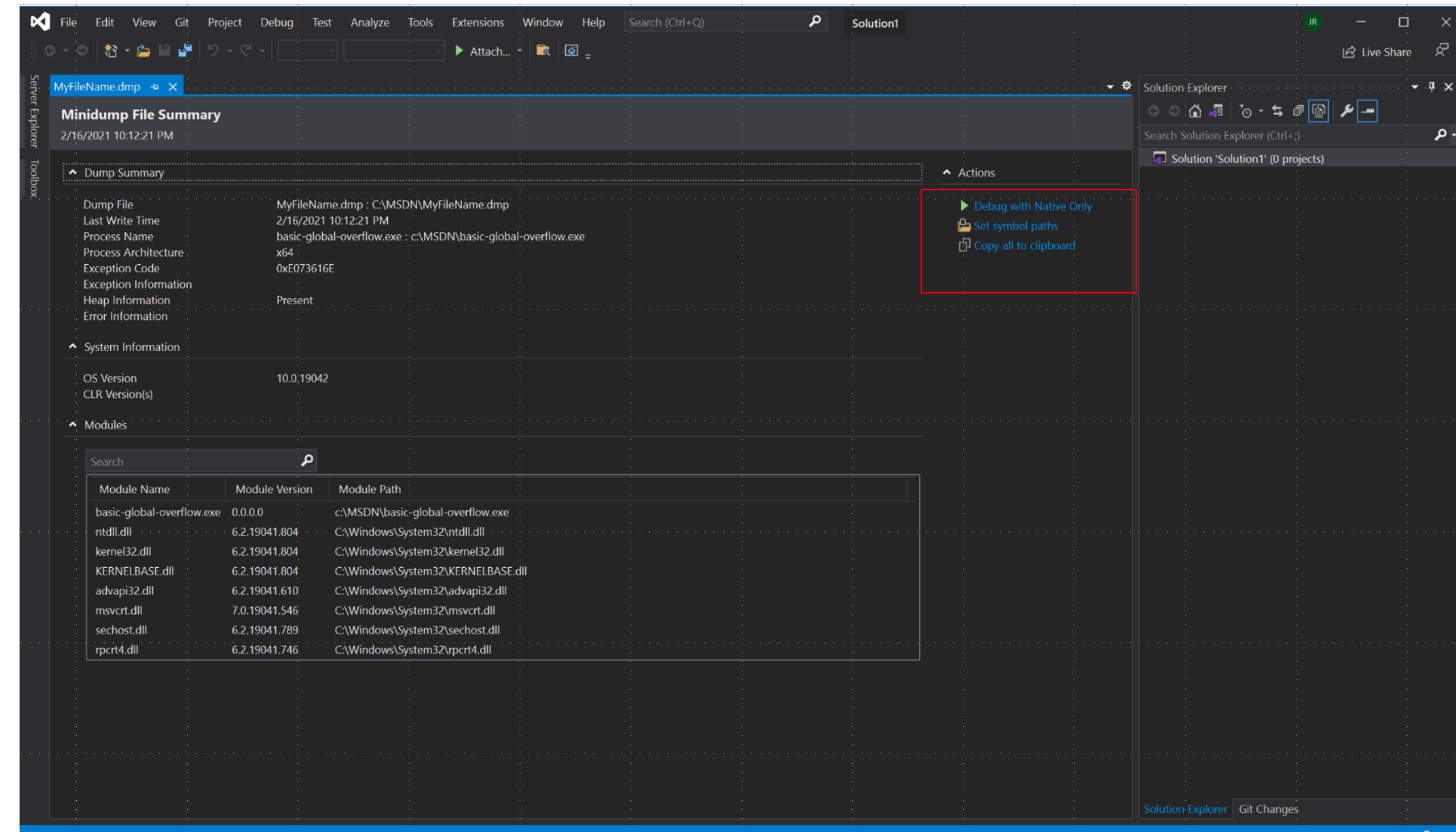
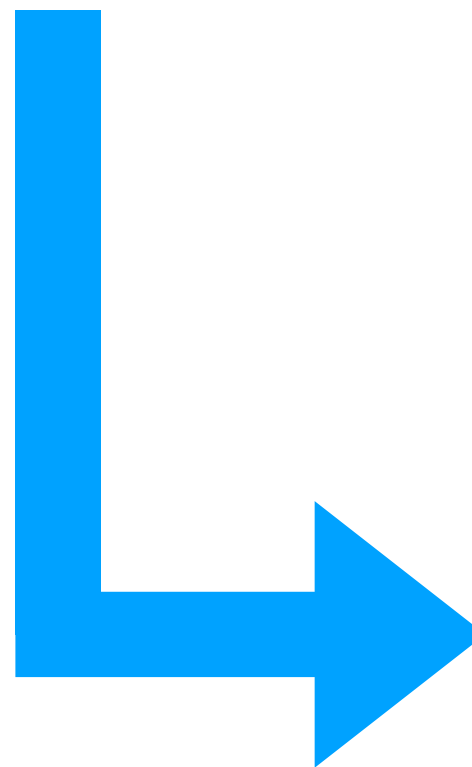


# ASAN cloud / distributed testing

You can create the **dump** on test or production infrastructure where the failure occurs, and debug it later on your **developer PC**

Crash dumps are created upon AddressSanitizer failures by setting the following environment variable:

```
set ASAN_SAVE_DUMPS=MyFileName.dmp
```



[docs.microsoft.com/en-us/cpp/sanitizers/asan-offline-crash-dumps](https://docs.microsoft.com/en-us/cpp/sanitizers/asan-offline-crash-dumps)



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

atlwin.h

```
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

CPU (% of all processors)

Summary Events Memory Usage CPU Usage

Events

Memory Usage

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 wcha

advinst.exe!ATL::CWindowImpl<ThemedPaneContainer,ATL::CWindow,ATL::CWinTraits<1442840576,0>>::Create(HWND\_\_ \* hWndParent, ATL::\_U\_REC

advinst.exe!WTL::CPaneContainerImpl<ThemedPaneContainer,ATL::CWindow,ATL::CWinTraits<1442840576,0>>::Create(HWND\_\_ \* hWndParent, unsi

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

advinst.exe!MsiFilesView::ProcessWindowMessage(HWND\_\_ \* hWnd, unsigned int uMsg, unsigned int wParam, long lParam, long & lResult, unsigned l

[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



# ASan Testing 🚗💨 Dieselgate style :)

```
int main() {  
    #ifdef __SANITIZE_ADDRESS__  
        printf("Address sanitizer enabled");  
    #else  
        printf("Address sanitizer not enabled");  
    #endif  
    return 1;  
}
```

```
__declspec(no_sanitize_address)  
void test1() {  
    int x[100];  
    x[100] = 5; // ASan exception not caught  
}
```

```
void test2() {  
    __declspec(no_sanitize_address) int x[100];  
    x[100] = 5; // ASan exception not caught  
}
```

```
__declspec(no_sanitize_address) int g[100];  
void test3() {  
    g[100] = 5; // ASan exception not caught  
}
```



Q & A

# Address Sanitizer on Windows

ACCU  
2021



@ciura\_victor

**Victor Ciura**  
Principal Engineer



**CAPHYON**