

Memory Safety: Static vs Dynamic Analysis

Open4Tech

January 19, 2022



@ciura_victor

Victor Ciura
Principal Engineer



CAPHYON

Abstract

Clang-tidy is the go-to assistant for most C/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.

Dynamic/runtime analysis, on the other hand, can catch more classes of memory vulnerabilities, but comes with its own costs.

Let's 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 to pinpoint the failure.

Ask questions as we go...

Q & A

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

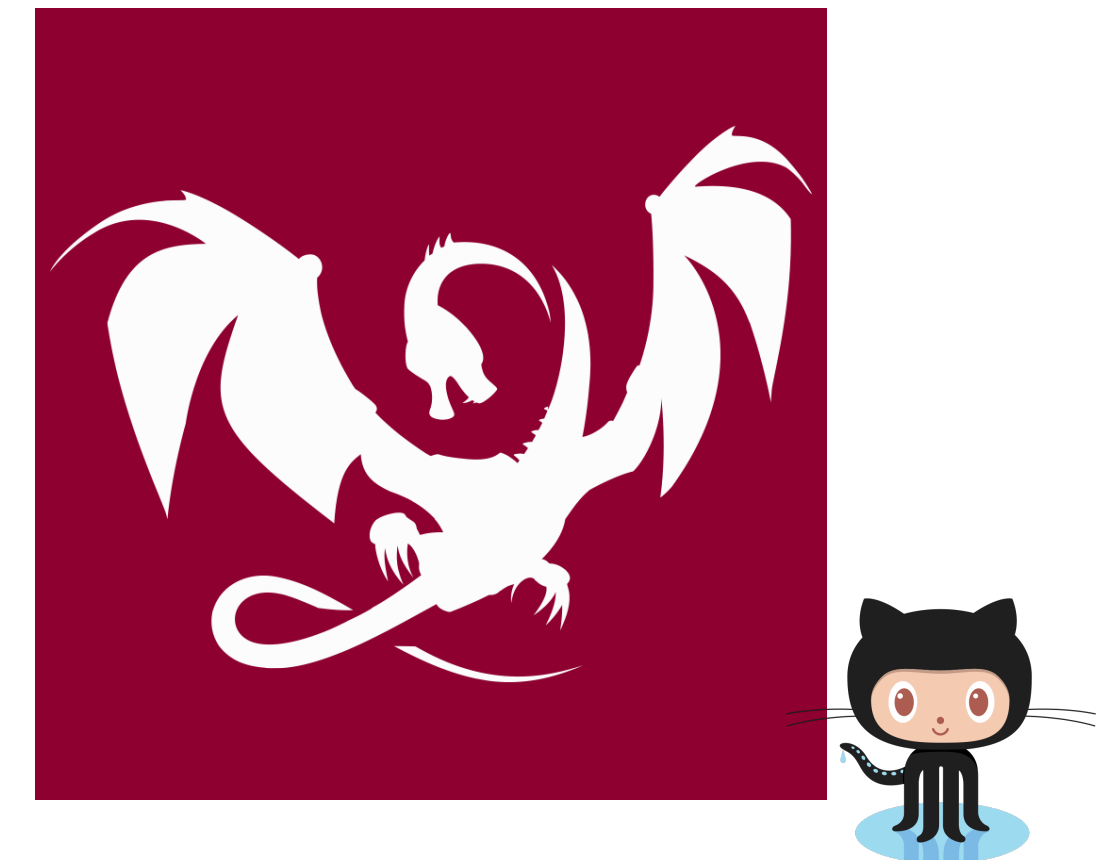
dynamic analyzer
(runtime)

+ fuzzing

I'm a tool maker



Advanced Installer



Clang Power Tools

Free/OSS

 **@ciura_victor**

Part I

Static Analysis



C++ Core Guidelines Checker



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

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

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



Static Analysis



Visual Studio integrates with:

- MSVC Code Analysis: <https://aka.ms/cpp/ca/bg>
- Clang Tidy: <https://aka.ms/cpp/clangtidy>
- VS Code Linters: <https://aka.ms/cpp/linter>

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

[youtube.com/watch?v=pQGRr4P16w](https://www.youtube.com/watch?v=pQGRr4P16w)



clang-tidy

300+ checks

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



clang-tidy

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



clang-tidy

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



clang-tidy

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

string checks

clang-tidy checks

Tidy Checks

Quick Search 🔍

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

Default Checks

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





clang-tidy bugprone-dangling-handle



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

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

Options:



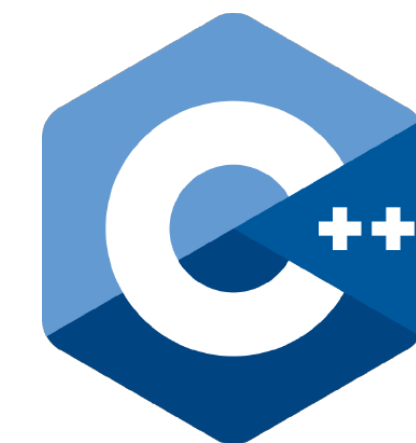
HandleClasses

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

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

Lifetime 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 safety: Preventing common dangling

`[-Wdangling-gsl]` diagnosed by default since **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 since **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>

Visual Studio 2019

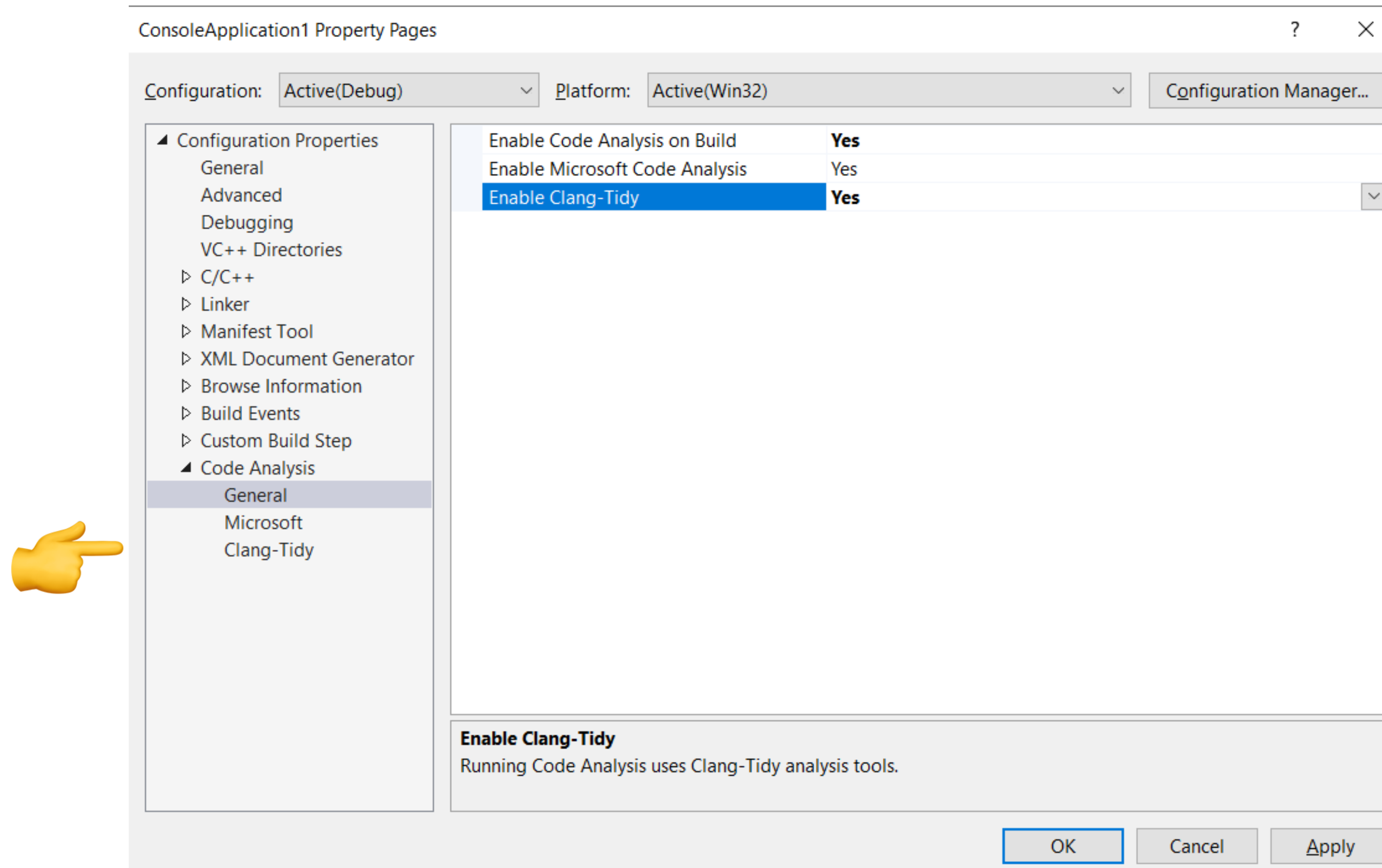
since **v16.4**

clang-tidy
code analysis



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

Visual Studio 2019/2022



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 2019/2022

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 2019/2022

clang-tidy warnings also display as in-editor squiggles

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

```
enum Positic
```

```
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



=



->



Free/OSS

Clang Power Tools

www.clangpowertools.com

LLVM

clang-tidy

clang++

clang-format

clang-check/query

Visual Studio

2017 / 2019 / 2022

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 8.1/10/11)

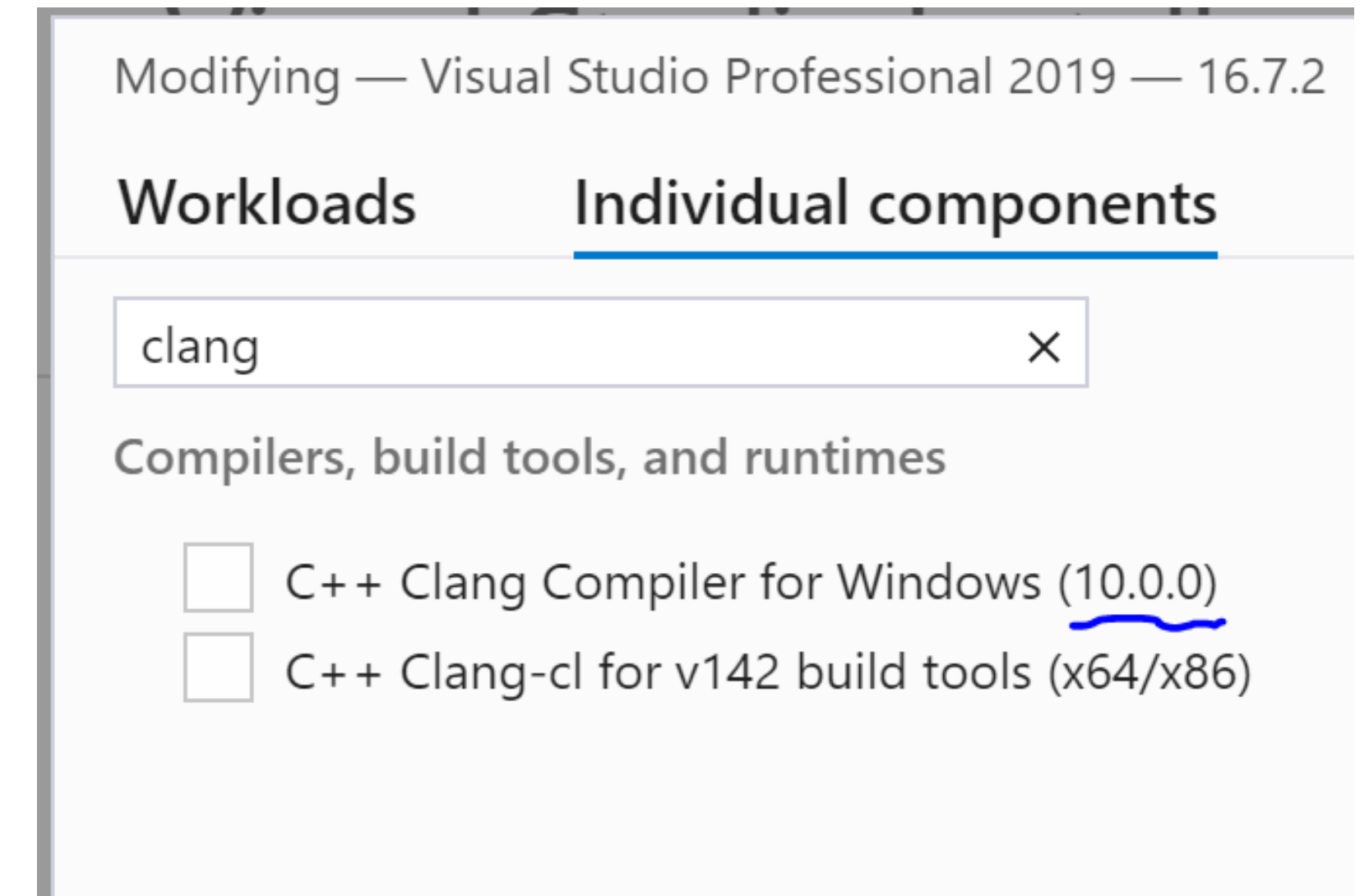
CFG is complementary to other exploit mitigations, such as:

- 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





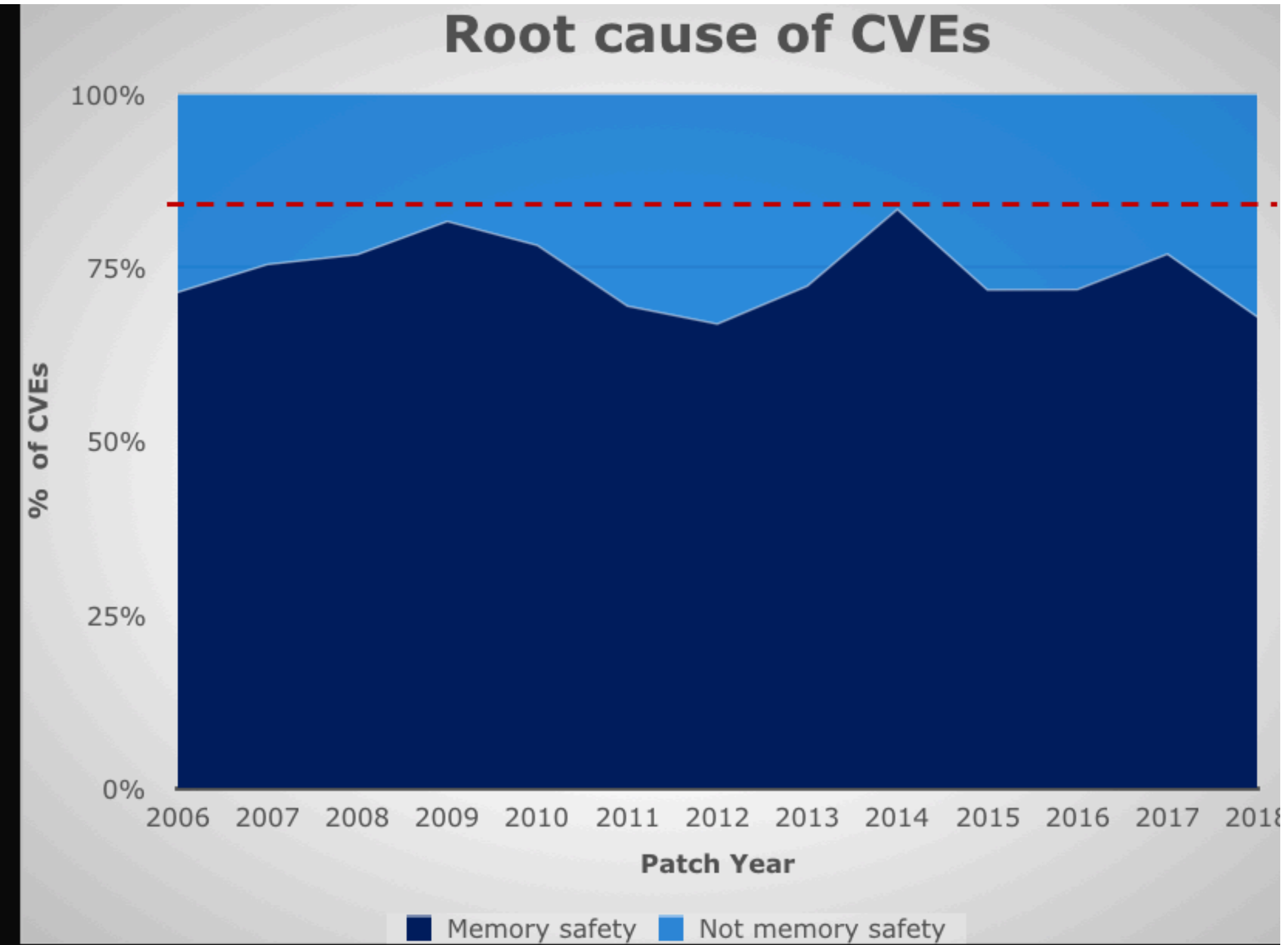
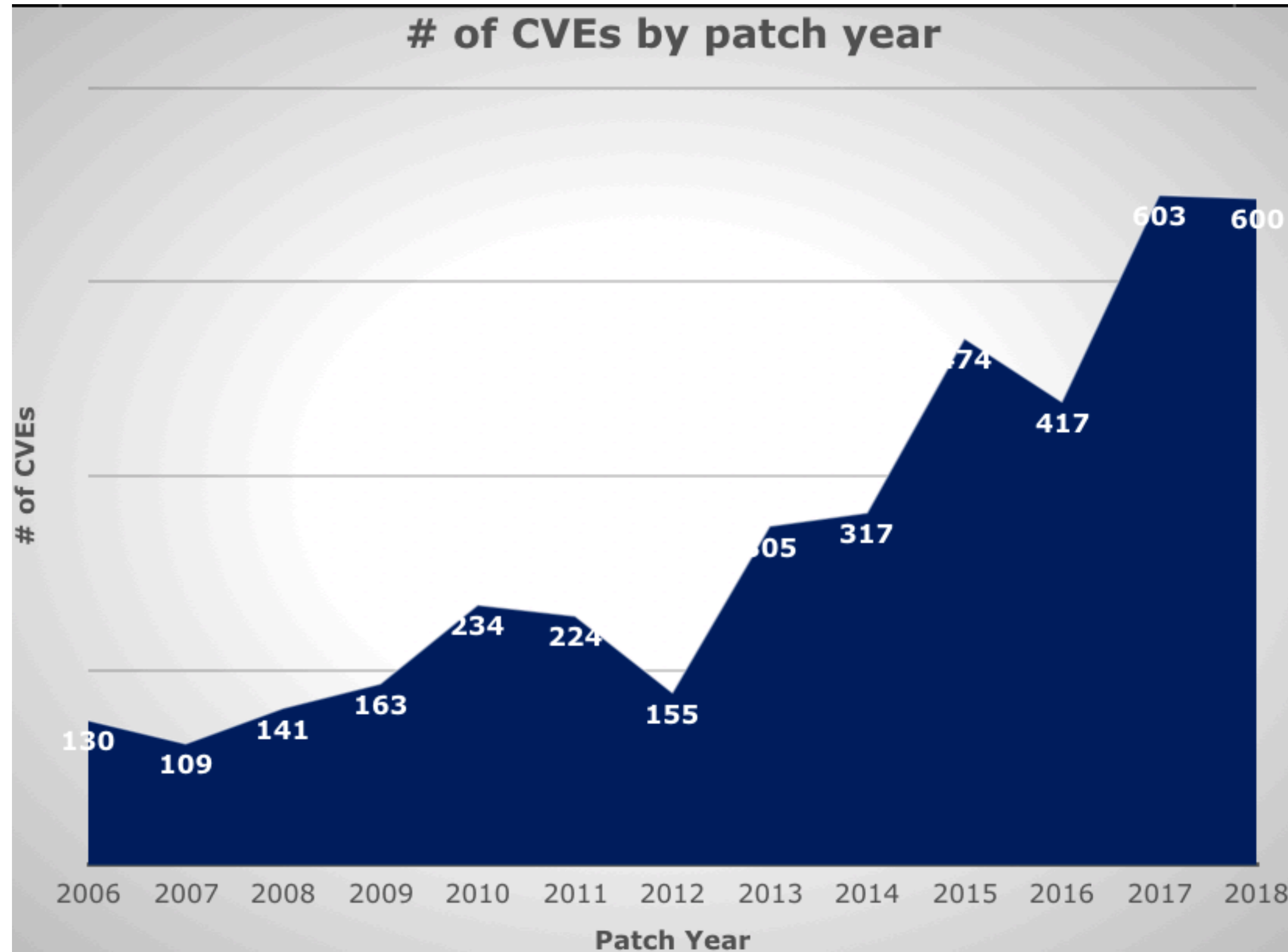
Sanitizers

- **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

Common Vulnerabilities and Exposures

Memory safety continues to dominate



youtube.com/watch?v=0EsqxGgYOQU



Address Sanitizer (ASan)

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

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

github.com/google/sanitizers/wiki/AddressSanitizer



Address Sanitizer (ASan)

Detects:

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

github.com/google/sanitizers/wiki/AddressSanitizer



Address Sanitizer (ASan)

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

github.com/google/sanitizers/wiki/AddressSanitizer



Address Sanitizer (ASan)

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

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

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

Visual Studio 2019

since **v16.4**

October 2019

Address Sanitizer (ASan)



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

Visual Studio 2019/2022

The screenshot shows the Visual Studio installation configuration window. The 'Workloads' tab is active, displaying a grid of workload options. Under 'Web & Cloud (4)', 'ASP.NET and web development' and 'Python development' are unchecked. Under 'Windows (3)', '.NET desktop development', 'Desktop development with C++', and 'Universal Windows Platform development' are checked. The 'Installation details' pane on the right shows the 'Desktop development with C++' workload selected, with 'C++ AddressSanitizer (Experimental)' highlighted by a red box and a yellow hand icon pointing to it. Other optional features like 'MSVC v142 - VS 2019 C++ x64/x86 build tools' and 'Windows 10 SDK' are also checked. The location is set to 'C:\Program Files (x86)\Microsoft Visual Studio\2019\Preview' and the total space required is 0 KB.

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.

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

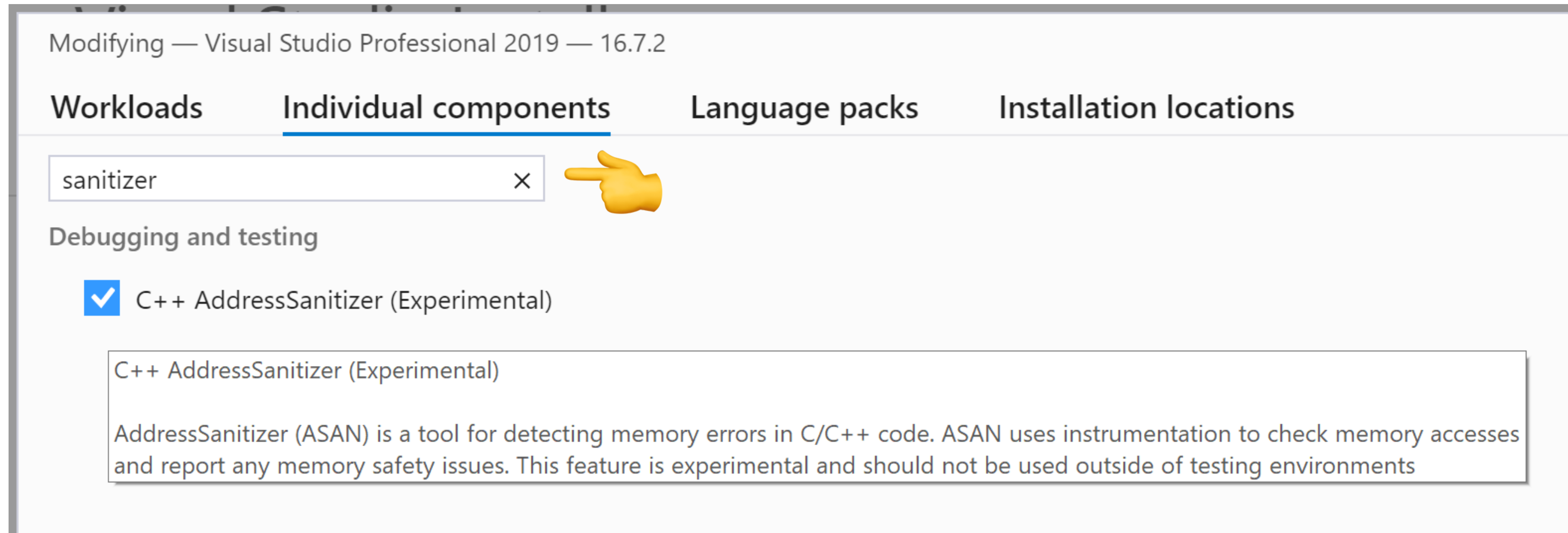
Total space required 0 KB

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 Close



Visual Studio 2019/2022



ASAN is out of Experimental since v16.9

Visual Studio 2019/2022

The screenshot shows the 'SystemScanner Property Pages' dialog box in Visual Studio. The 'Configuration' is set to 'Debug' and the 'Platform' is 'All Platforms'. The left sidebar shows the 'C/C++' section expanded to the 'General' tab. The main area displays a list of compiler options, with 'Enable Address Sanitizer (Experimental)' highlighted in blue and set to 'Yes (/fsanitize=address)'. Below the list, a summary box states: 'Enable Address Sanitizer (Experimental) Compiles and links program with AddressSanitizer. Currently available for x86 and x64 builds.' At the bottom right, there are 'OK', 'Cancel', and 'Apply' buttons.

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)

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

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

docs.microsoft.com/en-us/cpp/sanitizers/asan

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)

ASan features:

ASan features:

- expanded `RtlAllocateHeap` support

ASan features:

- expanded `RtlAllocateHeap` support
- support for the legacy `GlobalAlloc` and `LocalAlloc` family of memory functions

```
ASAN_OPTIONS=windows_hook_legacy_allocators=true
```

ASan features:

- expanded `RtlAllocateHeap` support
- 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

ASan features:

- expanded `RtlAllocateHeap` support
- 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

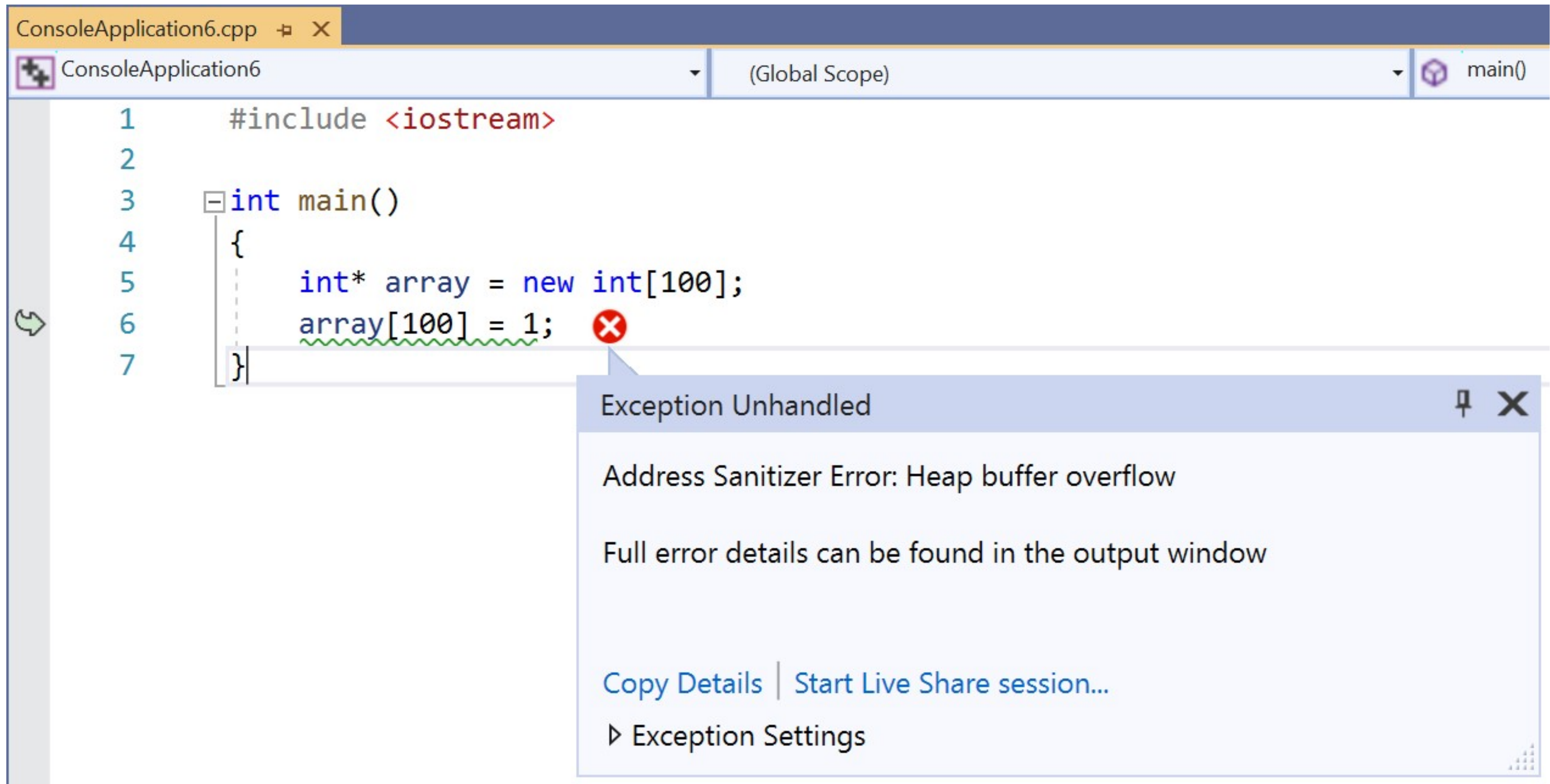
ASan features:

- expanded `RtlAllocateHeap` support
- 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

Address Sanitizer (ASan)



The screenshot shows a C++ IDE window titled "ConsoleApplication6.cpp" with a tab for "ConsoleApplication6". The code is at the "Global Scope" and "main()" level. The code is as follows:

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

Line 6, `array[100] = 1;`, is underlined with a red squiggly line and has a red 'X' icon next to it. A tooltip is displayed over this line with the following text:

Exception Unhandled

Address Sanitizer Error: Heap buffer overflow

Full error details can be found in the output window

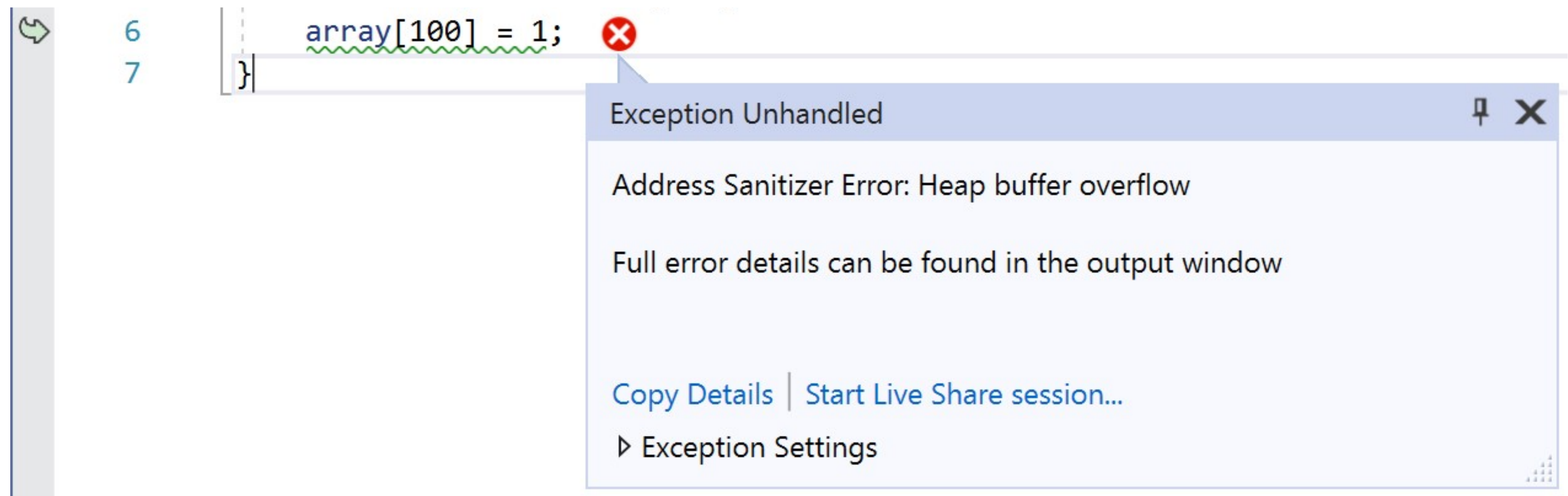
[Copy Details](#) | [Start Live Share session...](#)

▸ 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	Modul...
ShareSource.exe	1.0.0.0
ntdll.dll	10.0.177
kernel32.dll	10.0.177



Exception Unhandled
ASAN Error: Stack Buffer Overflow

```
109 CloseHandle(fileHandle);  
110  
111 void* freed_pointer = malloc  
112 free(freed_pointer); //we'1  
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 **
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 exception. The code includes a loop that writes to a stack-allocated array, followed by a double-free attempt. An "Exception Unhandled" dialog box is open, displaying "ASAN Error: Stack Buffer Overflow" and providing links to Azure Machine Learning buckets and a job submission page. A blue arrow points from the dialog to the output window, which shows memory dump data.

```

109     CloseHandle(FileHandle);
110
111     void* freed_pointer = malloc(1024);
112     free(freed_pointer); //we'll never get here either
113
114     if (array[0] == 'a') {
115         if (array[1] == 'b')
116             if (array[2] == 'c')
117                 if (array[3] == 'd')
118                     if (array[4] == 'e')
119                         if (array[5] == 'f')
120                             if (array[6] == 'g')
121                                 if (array[7] == 'h')
122                                     if (array[8] == 'i')
123                                         if (array[9] == 'j')
124                                             if (array[300] == 'X')
125                                                 printf("we'll never get here either");
126
127         if (array[11] == 'k' && array[38] == 'g' && array[100] == 'b')
128         {
129             *((int*)freed_pointer) = 0x1c0debad; //uaf
130         }
131     } else if (array[23] == '\xba')
132     {
133         free(freed_pointer); //double free
134     }
135
136     else if (strstr(array, "short"))
137     {
138         BYTE* byte_ptr = (BYTE*)malloc(1);
139         byte_ptr[0] = 0;
140     }
141 }

```

Exception Unhandled

ASAN Error: Stack Buffer Overflow

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

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

Locals

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

Output

Show output from: Debug

```

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

```

Snapshot Loaded

How does it work ?

ASan is just **Malware**,
used for Good 🤪

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:

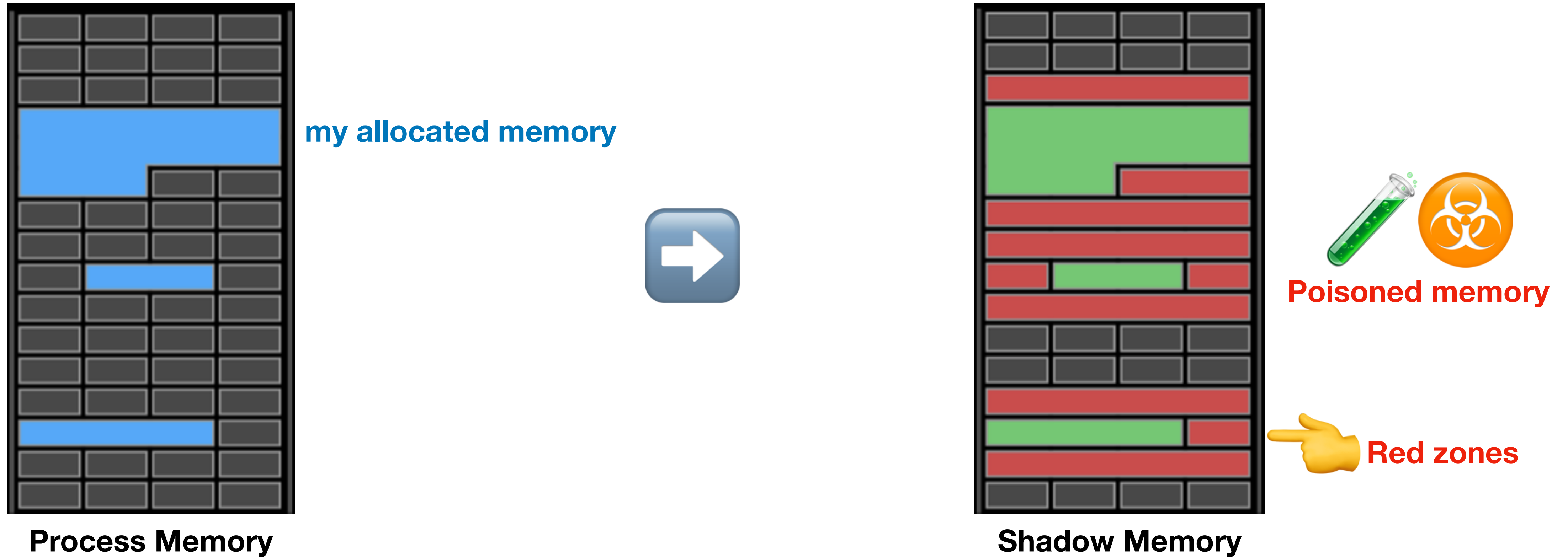
```
0x04d981eef0e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x04d981eef0f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x04d981ef0000: fa fa fa fa fa fa fa fa 00 00 00 00 00 00 00 00
=>0x04d981ef0030: 00 00 00 00 00 00 00 00 00 00 [fa] fa fa fa fa fa
0x04d981ef0040: fa fa fa fa fa fa fa 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)

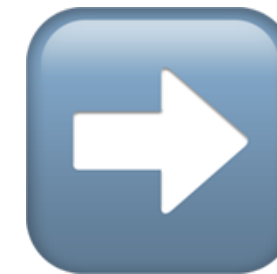
Shadow Mapping



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;  
}
```

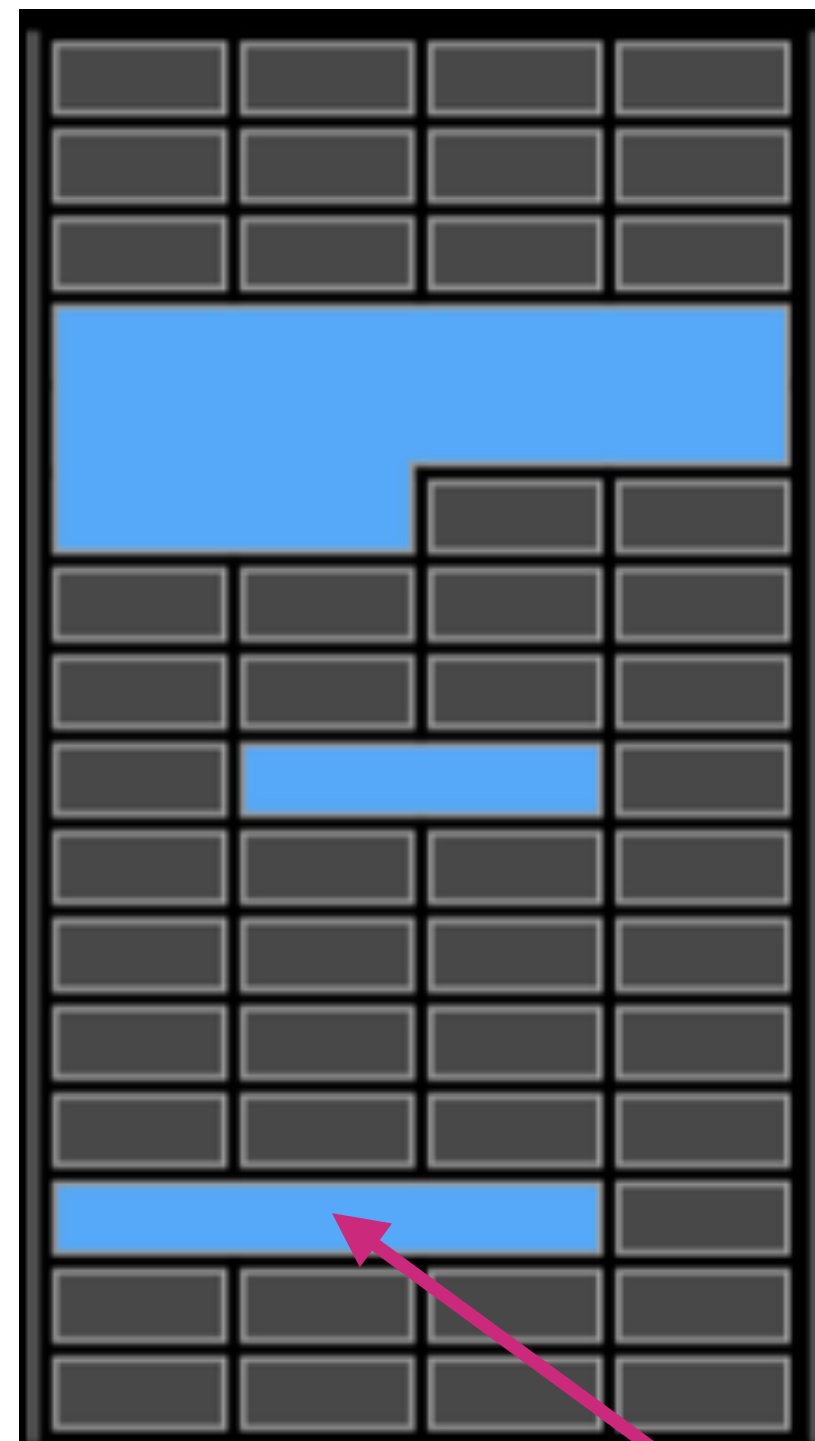
Location of shadow region in memory

A Shadow Byte:

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

Stack use after scope

Shadow Mapping

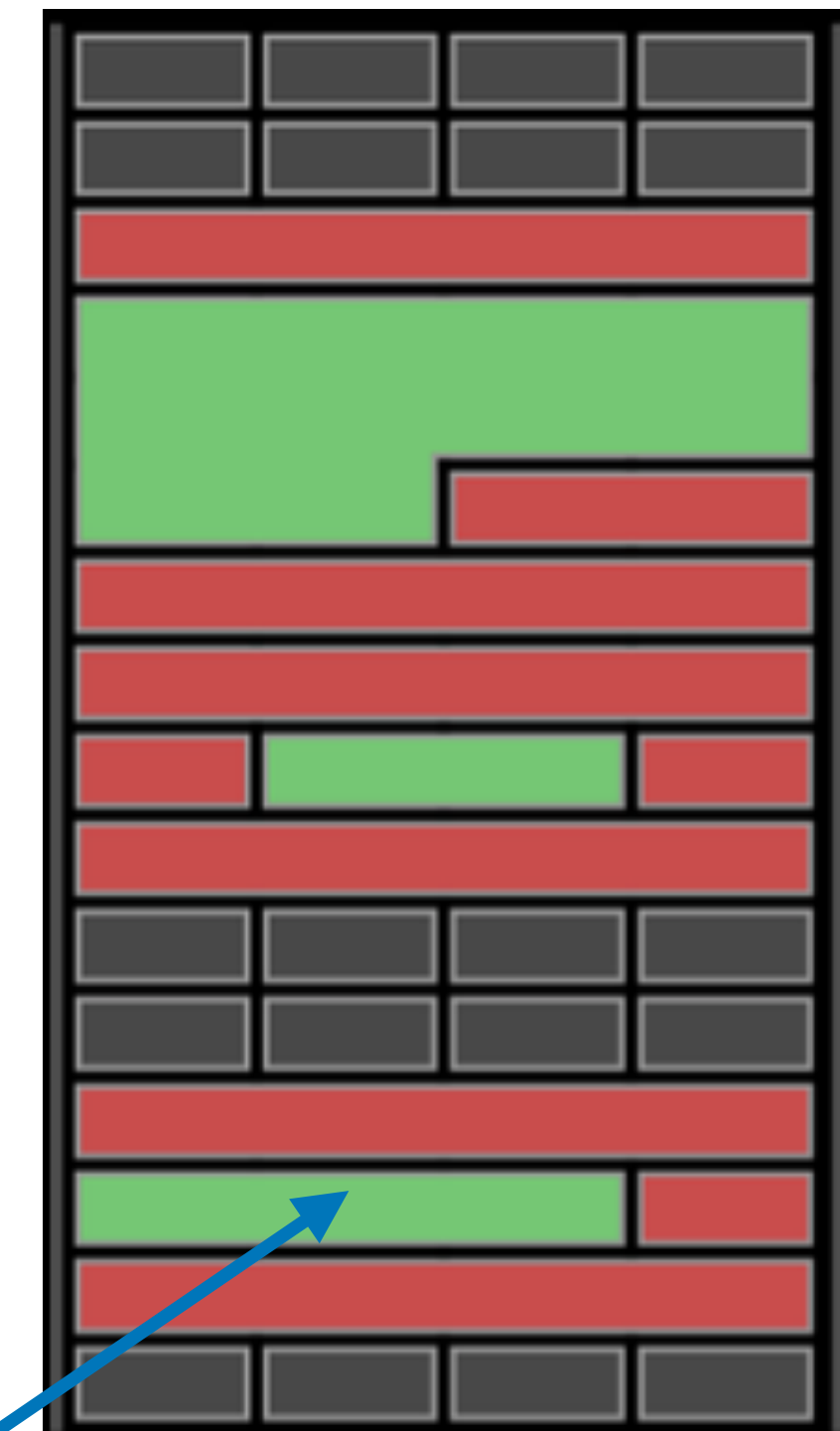


Process Memory

p

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

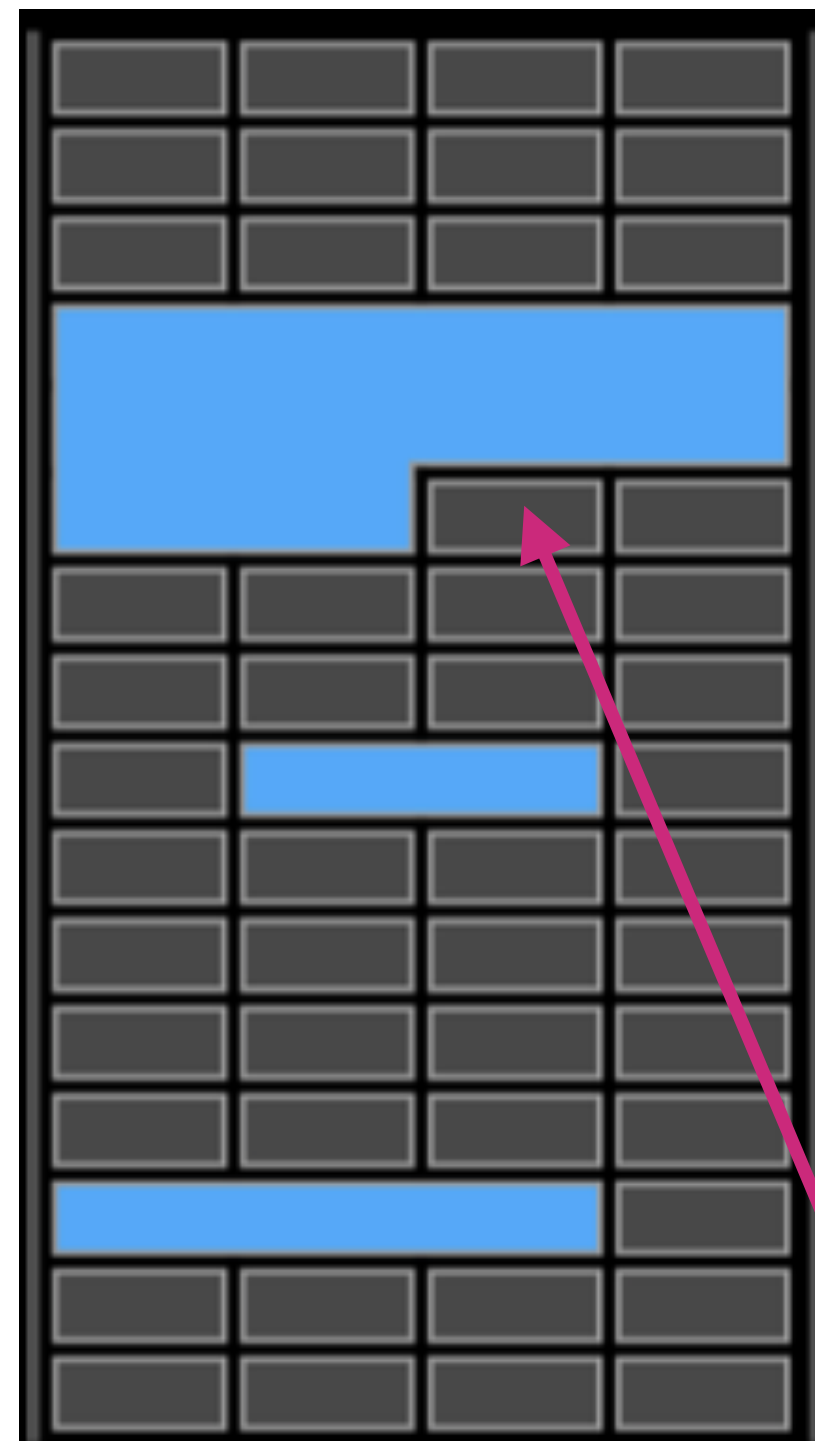
*p = 0xf00d



Shadow Memory

ShadowByte(p)

Shadow Mapping

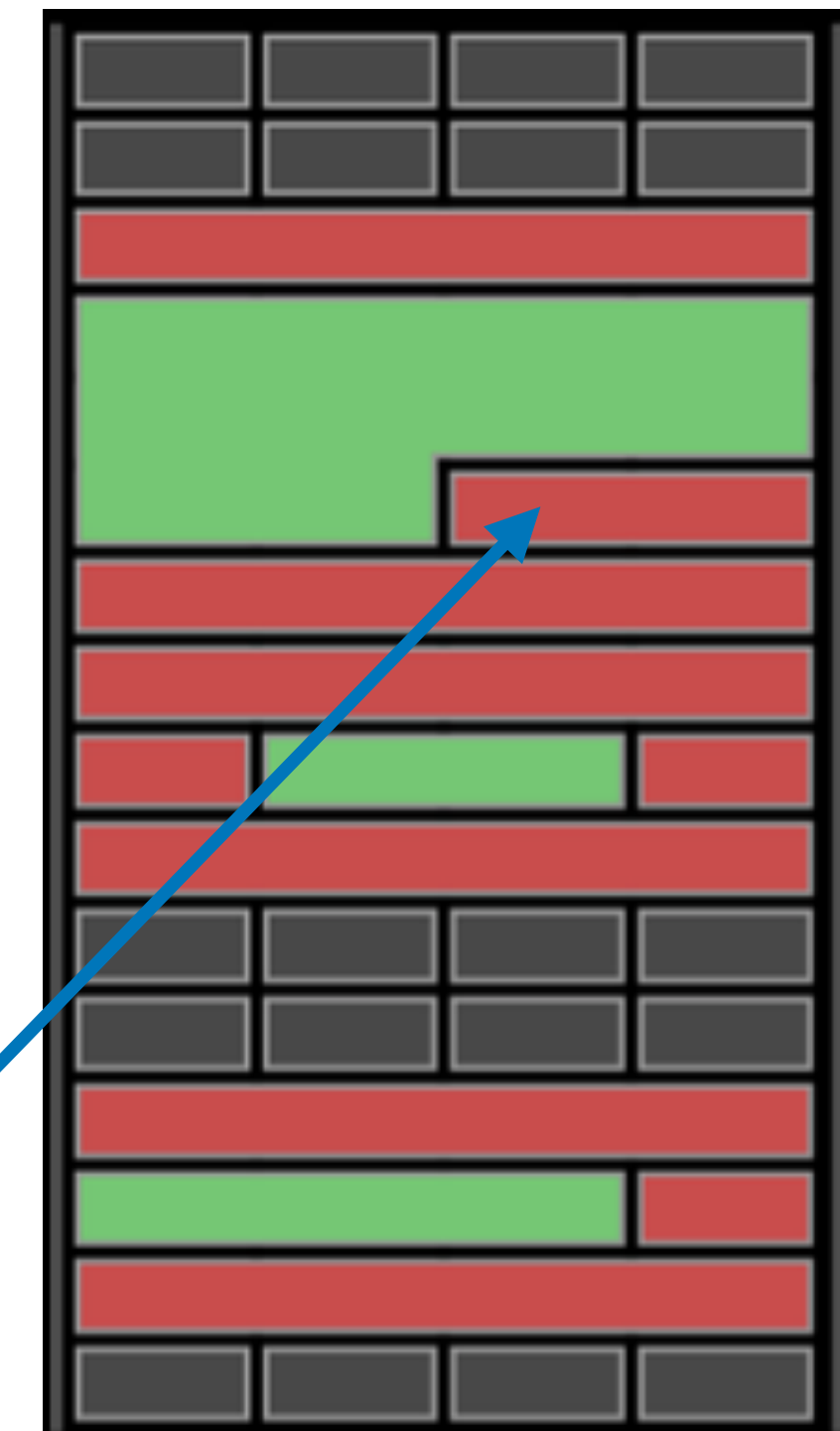


Process Memory

p

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

```
*p = 0xbadf00d
```



Shadow Memory

ShadowByte(p)

Heap Red Zones

malloc()



ASAN malloc()

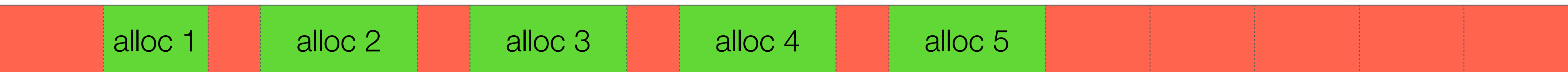


Heap Red Zones

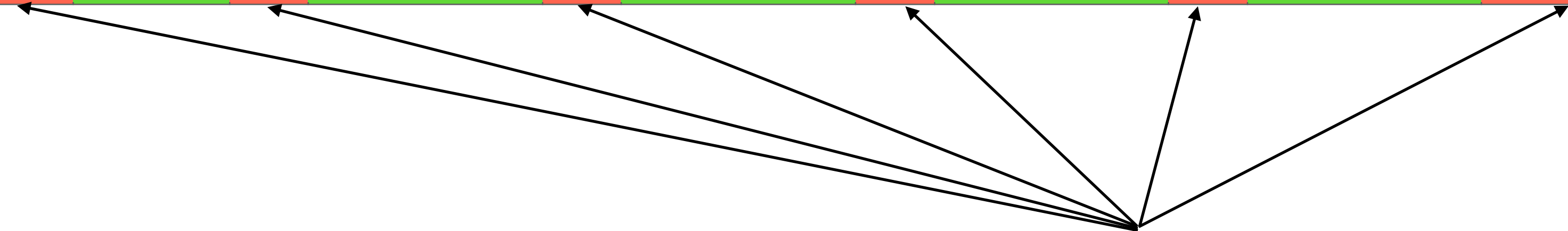
ASAN malloc()



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

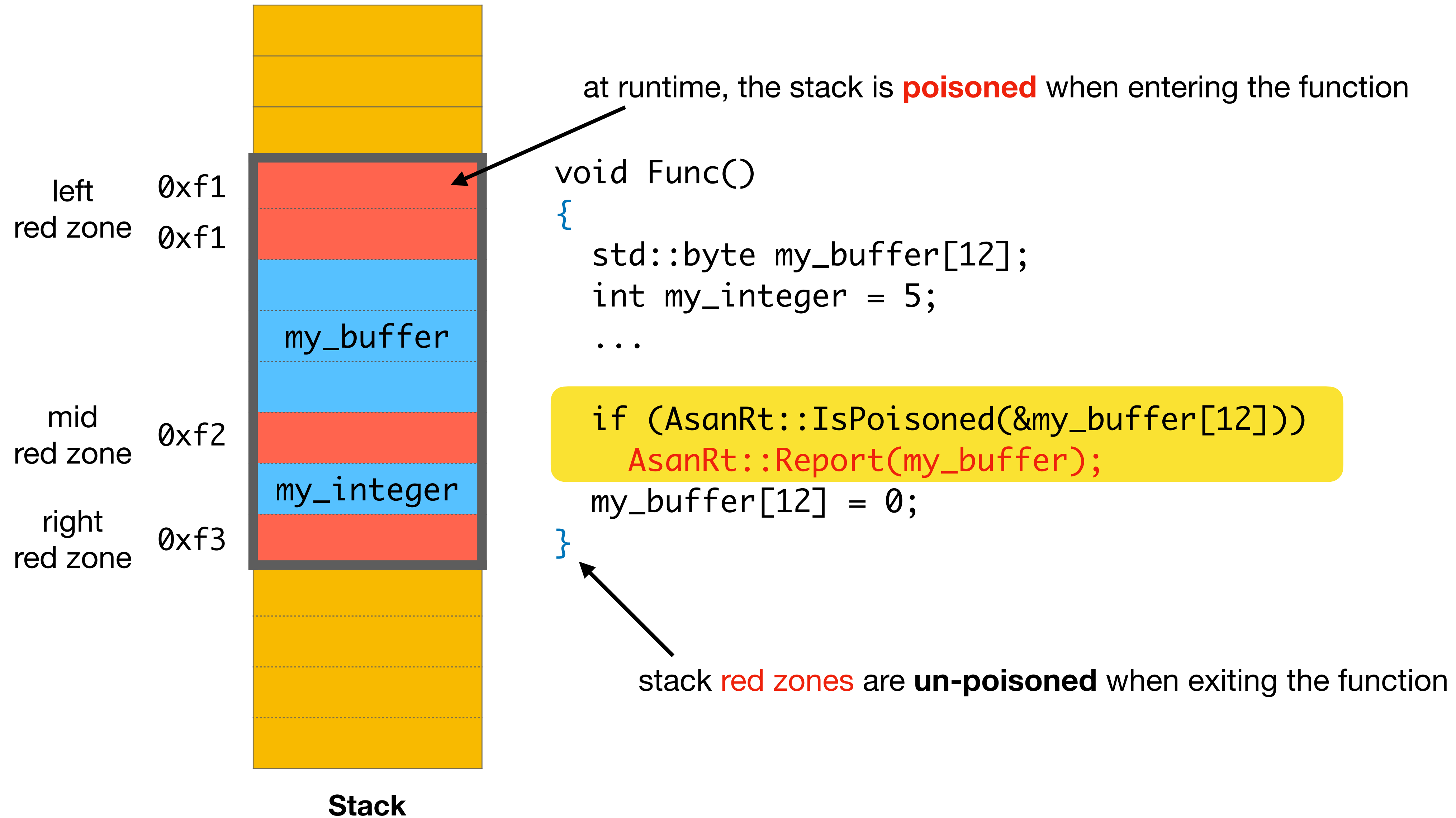
Stack Red Zones



Stack

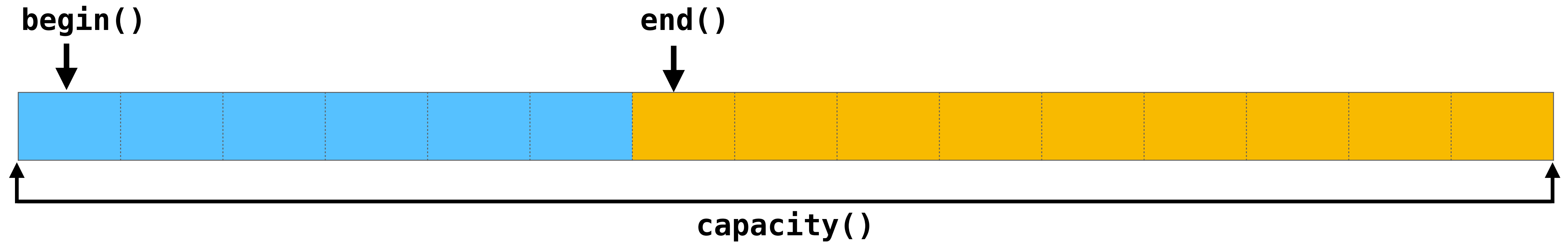
```
void Func()  
{  
    std::byte my_buffer[12];  
    int my_integer = 5;  
    ...  
    ...  
    ...  
    ...  
    my_buffer[12] = 0;  
}
```

Stack Red Zones



AddressSanitizer ContainerOverflow

`std::vector<T>`

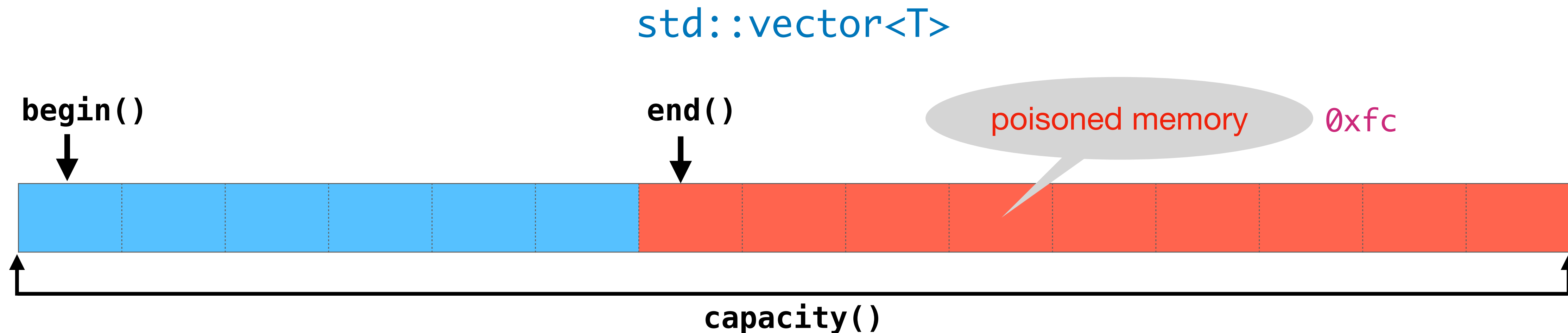


with the help of **code annotations** in `std::vector`

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

Problems & Gotchas

Stuff you need to know

`/ZI`

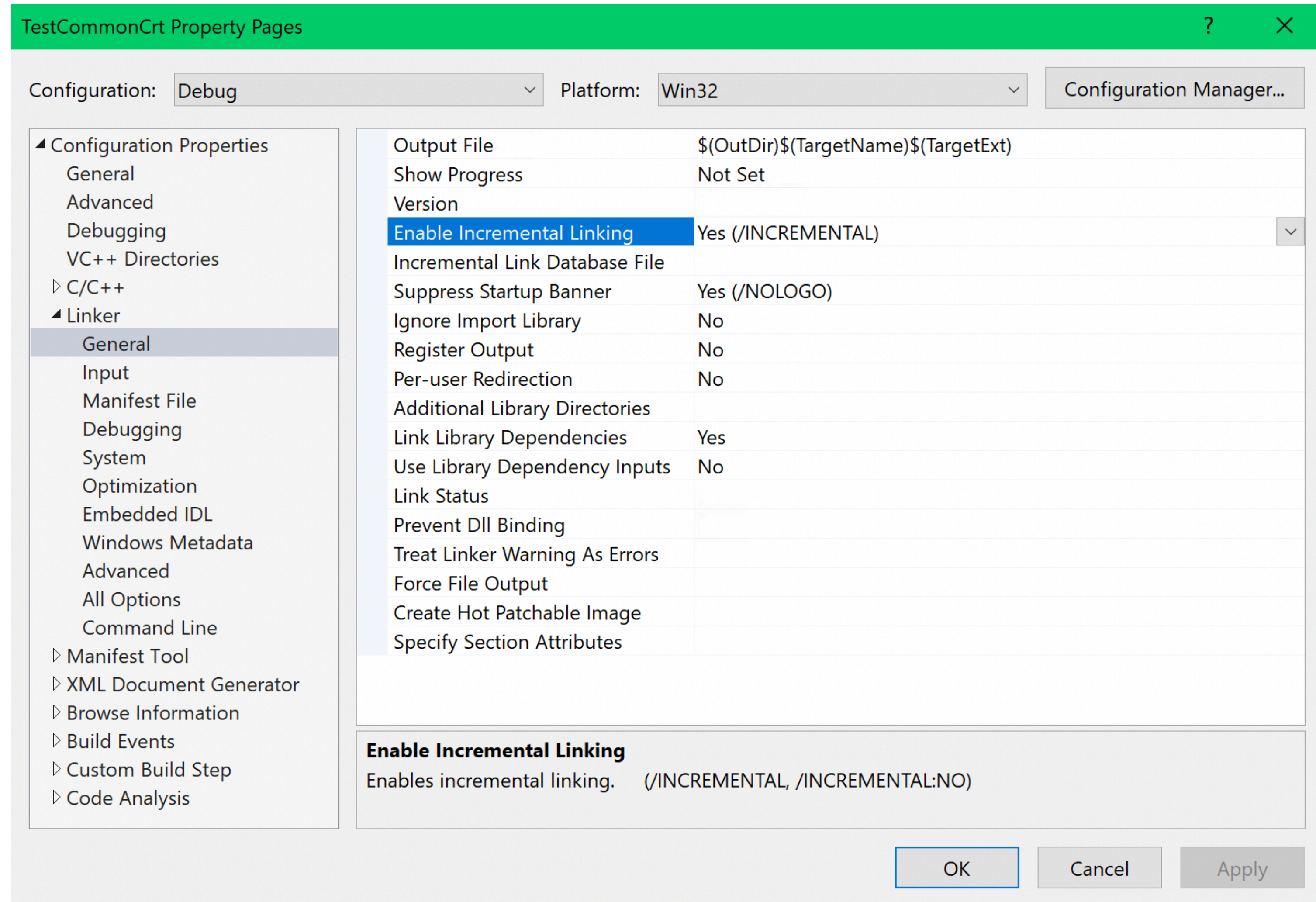
Edit and Continue (Debug)

error MSB8059:

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

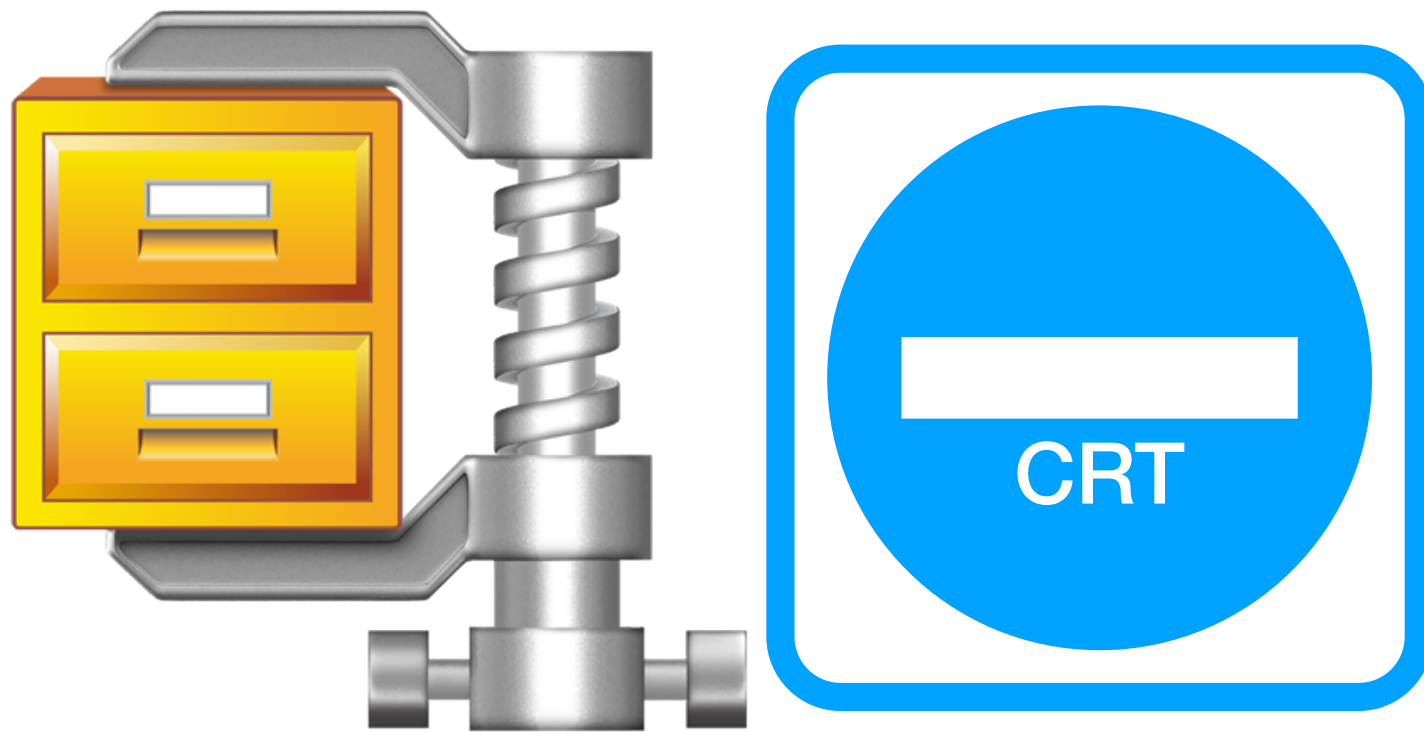
Link /INCREMENTAL

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

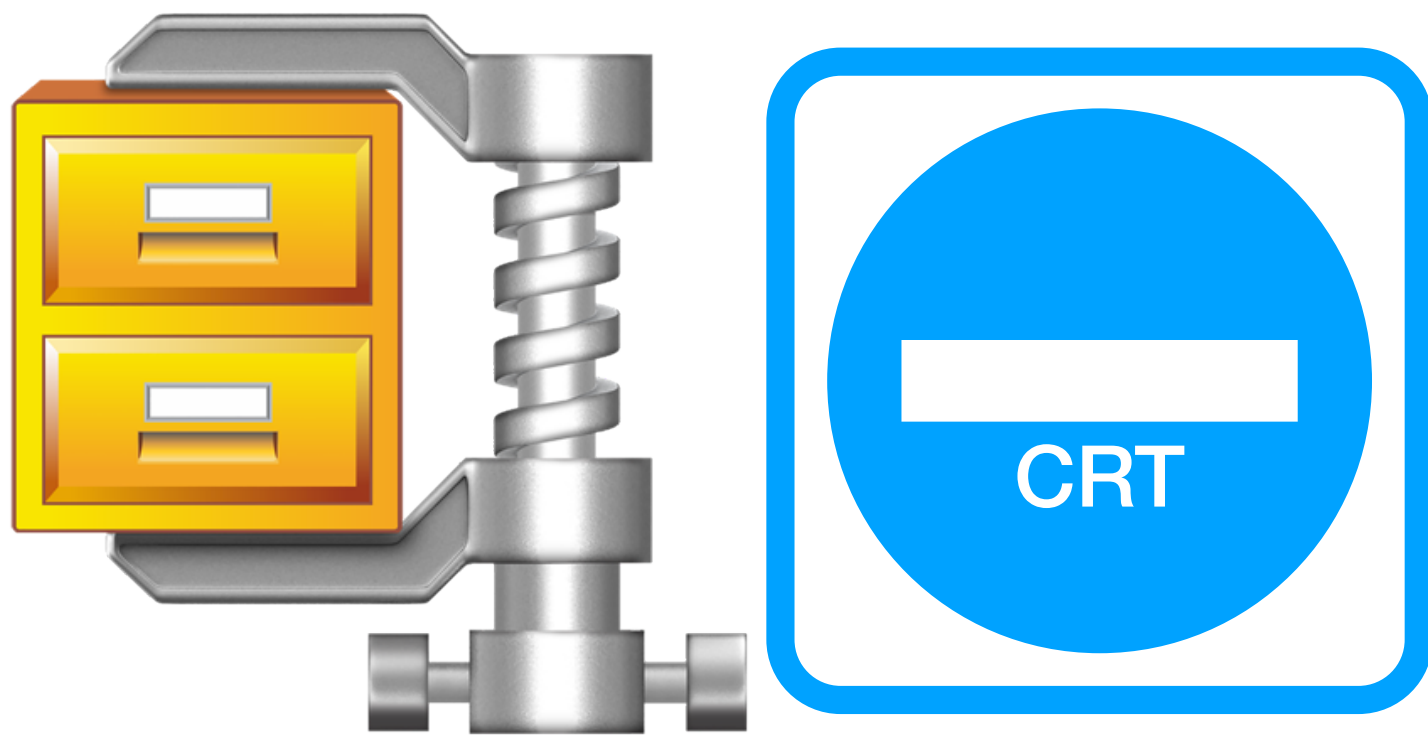
Configuration: All Configurations Platform: Win32 Configuration Manager...

- 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

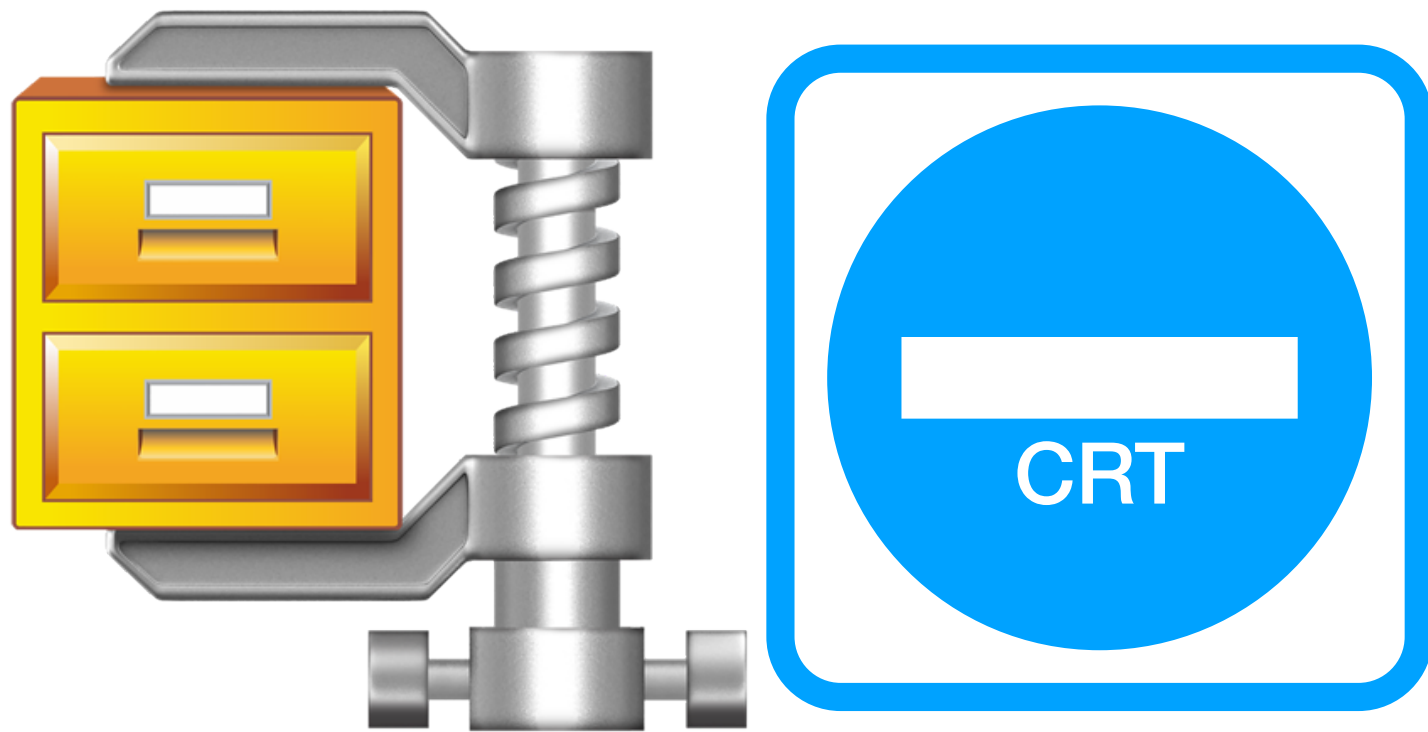


ASan + /NODEFAULTLIB

The linker will be very mad at you:

```
1>clang_rt.asan_dbg-i386.lib(ubsan_value.cc.obj) : error LNK2019: unresolved external symbol __allshl referenced in function "public: __int64 __thiscall __ubsan::Value::getSIntValue(void)const " (?getSIntValue@Value@__ubsan@QBE_JXZ)
1>clang_rt.asan_dbg-i386.lib(ubsan_value.cc.obj) : error LNK2019: unresolved external symbol __allshr referenced in function "public: __int64 __thiscall __ubsan::Value::getSIntValue(void)const " (?getSIntValue@Value@__ubsan@QBE_JXZ)
1>clang_rt.asan_dbg-i386.lib(ubsan_value.cc.obj) : error LNK2001: unresolved external symbol __fltused
1>clang_rt.asan_dbg-i386.lib(ubsan_diag.cc.obj) : error LNK2001: unresolved external symbol __fltused
1>clang_rt.asan_dbg-i386.lib(asan_malloc_win_moveable.cc.obj) : error LNK2001: unresolved external symbol __fltused
1>clang_rt.asan_dbg-i386.lib(asan_interceptors.cc.obj) : error LNK2001: unresolved external symbol __fltused
1>clang_rt.asan_dbg-i386.lib(ubsan_flags.cc.obj) : error LNK2019: unresolved external symbol _getenv referenced in function "char const * __cdecl __ubsan::GetFlag(char const *)" (?GetFlag@__ubsan@YAPBDPBDZ)
1>clang_rt.asan_dbg-i386.lib(ubsan_diag.cc.obj) : error LNK2019: unresolved external symbol ___stdio_common_vsprintf_s referenced in function _sprintf_s
1>clang_rt.asan_dbg-i386.lib(sanitizer_unwind_win.cc.obj) : error LNK2019: unresolved external symbol _memset referenced in function "private: void __thiscall __sanitizer::BufferedStackTrace::UnwindSlow(unsigned long,void *,unsigned int)" (?UnwindSlow@BufferedStackTrace@__sanitizer@YAHXXZ)
1>clang_rt.asan_dbg-i386.lib(interception_win.cc.obj) : error LNK2001: unresolved external symbol _memset
1>clang_rt.asan_dbg-i386.lib(asan_malloc_win_moveable.cc.obj) : error LNK2001: unresolved external symbol _memset
1>clang_rt.asan_dbg-i386.lib(sanitizer_interceptors.cc.obj) : error LNK2001: unresolved external symbol _memset
1>clang_rt.asan_dbg-i386.lib(sanitizer_symbolizer_win.cc.obj) : error LNK2019: unresolved external symbol _wcsrchr referenced in function "void __cdecl __sanitizer::InitializeDbgHelpIfNeeded(void)" (?InitializeDbgHelpIfNeeded@__sanitizer@YAHXXZ)
1>clang_rt.asan_dbg-i386.lib(sanitizer_symbolizer_win.cc.obj) : error LNK2019: unresolved external symbol _wcscat_s referenced in function "void __cdecl __sanitizer::InitializeDbgHelpIfNeeded(void)" (?InitializeDbgHelpIfNeeded@__sanitizer@YAHXXZ)
1>clang_rt.asan_dbg-i386.lib(sanitizer_allocator_checks.cc.obj) : error LNK2019: unresolved external symbol _errno referenced in function "void __cdecl __sanitizer::SetErrnoToENOMEM(void)" (?SetErrnoToENOMEM@__sanitizer@YAHXXZ)
1>clang_rt.asan_dbg-i386.lib(sanitizer_win.cc.obj) : error LNK2001: unresolved external symbol _errno
1>clang_rt.asan_dbg-i386.lib(asan_allocator.cc.obj) : error LNK2001: unresolved external symbol _errno
1>clang_rt.asan_dbg-i386.lib(sanitizer_win.cc.obj) : error LNK2019: unresolved external symbol _atexit referenced in function "int __cdecl __sanitizer::RunAtexit(void)" (?RunAtexit@__sanitizer@YAHXXZ)
1>clang_rt.asan_dbg-i386.lib(asan_malloc_win_moveable.cc.obj) : error LNK2001: unresolved external symbol _atexit
1>clang_rt.asan_dbg-i386.lib(sanitizer_win.cc.obj) : error LNK2019: unresolved external symbol _qsort referenced in function "void __cdecl __sanitizer::DumpProcessMap(void)" (?DumpProcessMap@__sanitizer@YAHXXZ)
1>clang_rt.asan_dbg-i386.lib(sanitizer_win.cc.obj) : error LNK2019: unresolved external symbol _alldiv referenced in function "unsigned __int64 __cdecl __sanitizer::NanoTime(void)" (?NanoTime@__sanitizer@YA_KXZ)
1>clang_rt.asan_dbg-i386.lib(sanitizer_win.cc.obj) : error LNK2019: unresolved external symbol _allmul referenced in function "unsigned __int64 __cdecl __sanitizer::NanoTime(void)" (?NanoTime@__sanitizer@YA_KXZ)
1>clang_rt.asan_dbg-i386.lib(sanitizer_win.cc.obj) : error LNK2019: unresolved external symbol _chkstk referenced in function "public: void __thiscall __sanitizer::ListOfModules::init(void)" (?init@ListOfModules@__sanitizer@QAEHXXZ)
1>clang_rt.asan_dbg-i386.lib(sanitizer_printf.cc.obj) : error LNK2019: unresolved external symbol _aullvrm referenced in function "int __cdecl __sanitizer::AppendNumber(char **,char const *,unsigned __int64,unsigned char,unsigned char)" (?AppendNumber@__sanitizer@YAHXXZ)
1>clang_rt.asan_dbg-i386.lib(interception_win.cc.obj) : error LNK2019: unresolved external symbol _stricmp referenced in function "bool __cdecl __interception::OverrideImportedFunction(char const *,char const *,char const *,unsigned long)" (?OverrideImportedFunction@__interception@YAHXXZ)
1>clang_rt.asan_dbg-i386.lib(asan_win.cc.obj) : error LNK2019: unresolved external symbol _except_handler3 referenced in function "void __cdecl __asan::InitializePlatformInterceptors(void)" (?InitializePlatformInterceptors@__asan@YAHXXZ)
1>clang_rt.asan_dbg-i386.lib(asan_win.cc.obj) : error LNK2019: unresolved external symbol _except_handler4 referenced in function "void __cdecl __asan::InitializePlatformInterceptors(void)" (?InitializePlatformInterceptors@__asan@YAHXXZ)
1>libvcasand.lib(vcasan.obj) : error LNK2001: unresolved external symbol _except_handler4
1>clang_rt.asan_dbg-i386.lib(asan_win.cc.obj) : error LNK2019: unresolved external symbol _tls_array referenced in function "void * __cdecl __asan::AsanTSDGet(void)" (?AsanTSDGet@__asan@YAPAXXZ)
1>clang_rt.asan_dbg-i386.lib(asan_win.cc.obj) : error LNK2019: unresolved external symbol _tls_index referenced in function "void * __cdecl __asan::AsanTSDGet(void)" (?AsanTSDGet@__asan@YAPAXXZ)
1>clang_rt.asan_dbg-i386.lib(asan_rtl.cc.obj) : error LNK2019: unresolved external symbol _aullrem referenced in function "void __cdecl __asan::InitializeHighMemEnd(void)" (?InitializeHighMemEnd@__asan@YAHXXZ)
1>clang_rt.asan_dbg-i386.lib(asan_malloc_win_moveable.cc.obj) : error LNK2019: unresolved external symbol _invalid_parameter referenced in function "void * __cdecl std::_Allocate_manually_vector_aligned<struct std::_Default_allocate_traits>" (?Allocate_manually_vector_aligned@std@YAHXXZ)
1>clang_rt.asan_dbg-i386.lib(asan_malloc_win_moveable.cc.obj) : error LNK2019: unresolved external symbol _CrtDbgReport referenced in function "void * __cdecl std::_Allocate_manually_vector_aligned<struct std::_Default_allocate_traits>" (?Allocate_manually_vector_aligned@std@YAHXXZ)
1>clang_rt.asan_dbg-i386.lib(asan_malloc_win_moveable.cc.obj) : error LNK2019: unresolved external symbol "public: __thiscall std::_Lockit::_Lockit(int)" (?_Lockit@std@QAEH@Z) referenced in function "public: __thiscall std::_List_iterator<struct std::_Default_allocate_traits>" (?_List_iterator@std@YAHXXZ)
1>clang_rt.asan_dbg-i386.lib(asan_malloc_win_moveable.cc.obj) : error LNK2019: unresolved external symbol "public: __thiscall std::_Lockit::~_Lockit(void)" (?_Lockit@std@QAEH@Z) referenced in function "public: __thiscall std::_List_iterator<struct std::_Default_allocate_traits>" (?_List_iterator@std@YAHXXZ)
1>clang_rt.asan_dbg-i386.lib(asan_malloc_win_moveable.cc.obj) : error LNK2019: unresolved external symbol ___std_exception_copy referenced in function "public: __thiscall std::bad_alloc::bad_alloc(class std::bad_alloc const &)" (?bad_alloc@std@YAHXXZ)
1>clang_rt.asan_dbg-i386.lib(asan_malloc_win_moveable.cc.obj) : error LNK2019: unresolved external symbol ___std_exception_destroy referenced in function "public: virtual __thiscall std::bad_array_new_length::~bad_array_new_length(void)" (?bad_array_new_length@std@YAHXXZ)
1>clang_rt.asan_dbg-i386.lib(asan_malloc_win_moveable.cc.obj) : error LNK2019: unresolved external symbol "void __cdecl std::_Xlength_error(char const *)" (?_Xlength_error@std@YAPBD@Z) referenced in function "protected: struct std::pair<struct std::_Default_allocate_traits,struct std::_Default_allocate_traits>" (?pair@std@YAHXXZ)
1>clang_rt.asan_dbg-i386.lib(asan_malloc_win_moveable.cc.obj) : error LNK2019: unresolved external symbol "void __cdecl std::_Xout_of_range(char const *)" (?_Xout_of_range@std@YAPBD@Z) referenced in function "public: void * __thiscall std::_List_iterator<struct std::_Default_allocate_traits>" (?_List_iterator@std@YAHXXZ)
1>clang_rt.asan_dbg-i386.lib(asan_malloc_win_moveable.cc.obj) : error LNK2019: unresolved external symbol _Mtx_init_in_situ referenced in function "void __cdecl 'dynamic initializer for 'GlobalHeapMutex''(void)" (?_EGlobalHeapMutex@std@YAHXXZ)
1>clang_rt.asan_dbg-i386.lib(asan_malloc_win_moveable.cc.obj) : error LNK2019: unresolved external symbol _Mtx_destroy_in_situ referenced in function "void __cdecl 'dynamic atexit destructor for 'GlobalHeapMutex''(void)" (?_FGlobalHeapMutex@std@YAHXXZ)
1>clang_rt.asan_dbg-i386.lib(asan_malloc_win_moveable.cc.obj) : error LNK2019: unresolved external symbol _Mtx_lock referenced in function "public: void * __thiscall MoveableMemoryManager::Alloc(unsigned long,unsigned int)" (?Alloc@MoveableMemoryManager@__asan@YAHXXZ)
1>clang_rt.asan_dbg-i386.lib(asan_malloc_win_moveable.cc.obj) : error LNK2019: unresolved external symbol _Mtx_unlock referenced in function "public: void * __thiscall MoveableMemoryManager::Alloc(unsigned long,unsigned int)" (?Alloc@MoveableMemoryManager@__asan@YAHXXZ)
1>clang_rt.asan_dbg-i386.lib(asan_malloc_win_moveable.cc.obj) : error LNK2019: unresolved external symbol "void __cdecl std::_Throw_C_error(int)" (?_Throw_C_error@std@YAHX@Z) referenced in function "public: void * __thiscall MoveableMemoryManager::Alloc(unsigned long,unsigned int)" (?Alloc@MoveableMemoryManager@__asan@YAHXXZ)
1>clang_rt.asan_dbg-i386.lib(asan_malloc_win_moveable.cc.obj) : error LNK2019: unresolved external symbol _CxxThrowException@8 referenced in function __catch$??$_Emplace_reallocate@ABQAVMovableAllocEntry@@@?vector@PAVMovableAllocEntry@@@YAHXXZ
1>clang_rt.asan_dbg-i386.lib(asan_malloc_win_moveable.cc.obj) : error LNK2019: unresolved external symbol ___CxxFrameHandler3 referenced in function __ehandler$??$_Emplace_reallocate@ABQAVMovableAllocEntry@@@?vector@PAVMovableAllocEntry@@@YAHXXZ
1>libvcasand.lib(vcasan.obj) : error LNK2001: unresolved external symbol ___CxxFrameHandler3
1>clang_rt.asan_dbg-i386.lib(asan_malloc_win_moveable.cc.obj) : error LNK2019: unresolved external symbol _ftoui3 referenced in function "protected: struct std::pair<struct std::_List_node<struct std::pair<void * const,class MoveableAllocEntry>>>" (?pair@std@YAHXXZ)
1>clang_rt.asan_dbg-i386.lib(asan_malloc_win_moveable.cc.obj) : error LNK2019: unresolved external symbol _ceil referenced in function "protected: struct std::pair<struct std::_List_node<struct std::pair<void * const,class MoveableAllocEntry>>>" (?pair@std@YAHXXZ)
1>clang_rt.asan_dbg-i386.lib(asan_malloc_win_moveable.cc.obj) : error LNK2019: unresolved external symbol _memcpy referenced in function "public: void * __thiscall MoveableMemoryManager::ReallocFixedToHandle(void *,bool)" (?ReallocFixedToHandle@MoveableMemoryManager@__asan@YAHXXZ)
1>clang_rt.asan_dbg-i386.lib(asan_interceptors.cc.obj) : error LNK2001: unresolved external symbol _memcpy
1>libvcasand.lib(vcasan.obj) : error LNK2001: unresolved external symbol _memcpy
1>clang_rt.asan_dbg-i386.lib(asan_malloc_win_moveable.cc.obj) : error LNK2019: unresolved external symbol _memmove referenced in function "class MoveableAllocEntry * * * __cdecl std::_Copy_memmove<class MoveableAllocEntry * * *,class MoveableAllocEntry * * *,class MoveableAllocEntry * * *,class MoveableAllocEntry * * *>" (?Copy_memmove@std@YAHXXZ)
```

ASan runtime assumes
CRT is linked



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

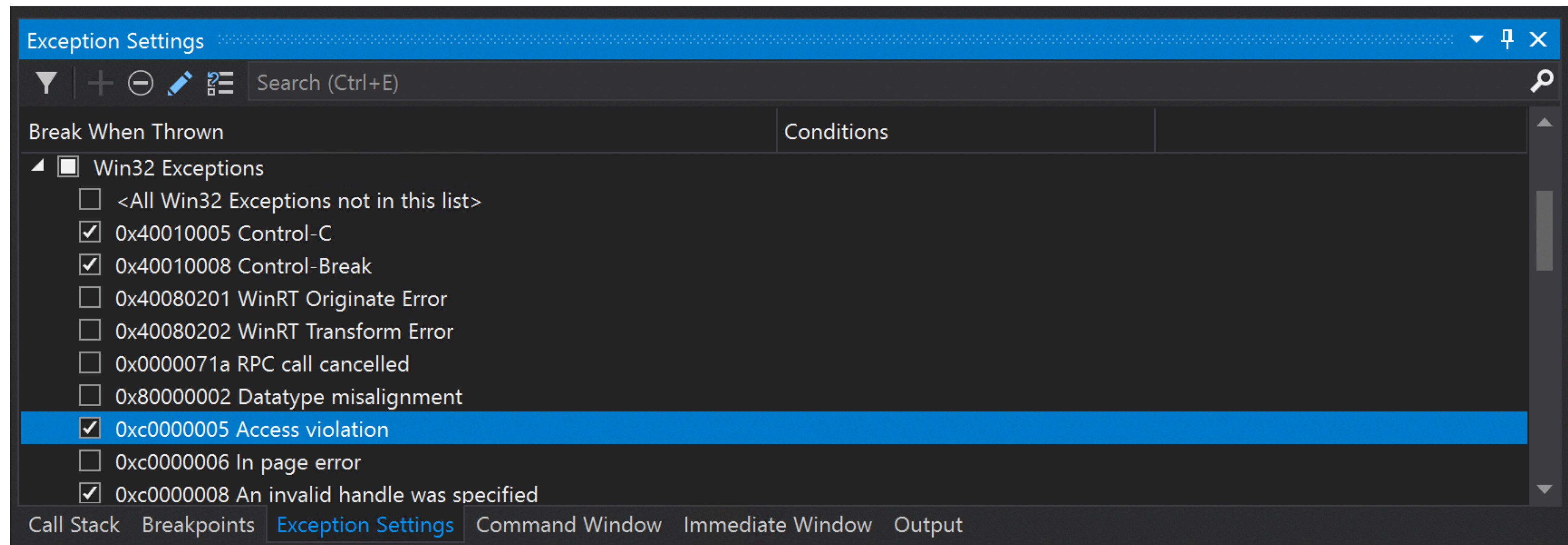
Access Violation Exceptions

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/

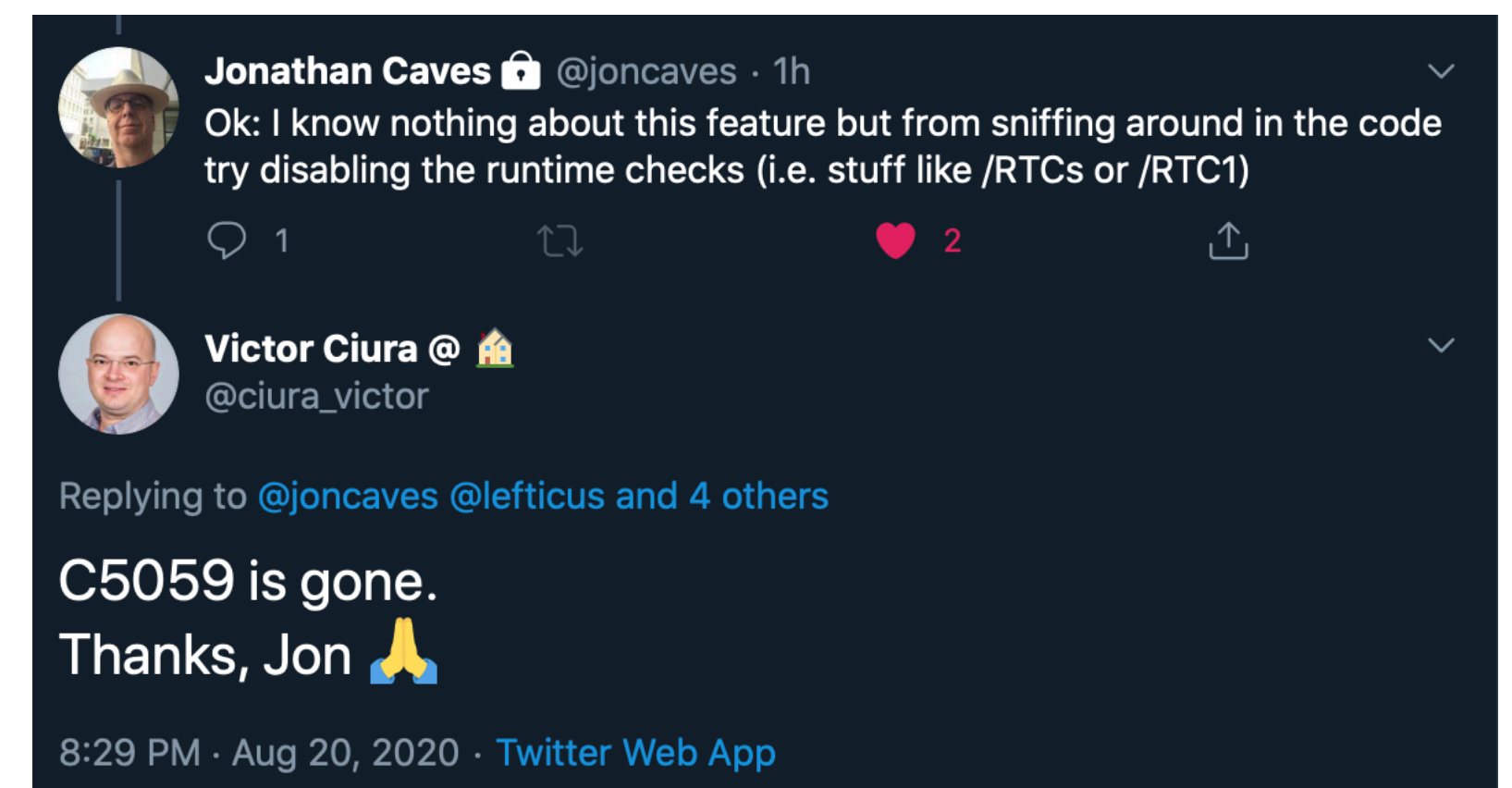
/RTCs and /RTC1 Runtime Checks

warning C5059:

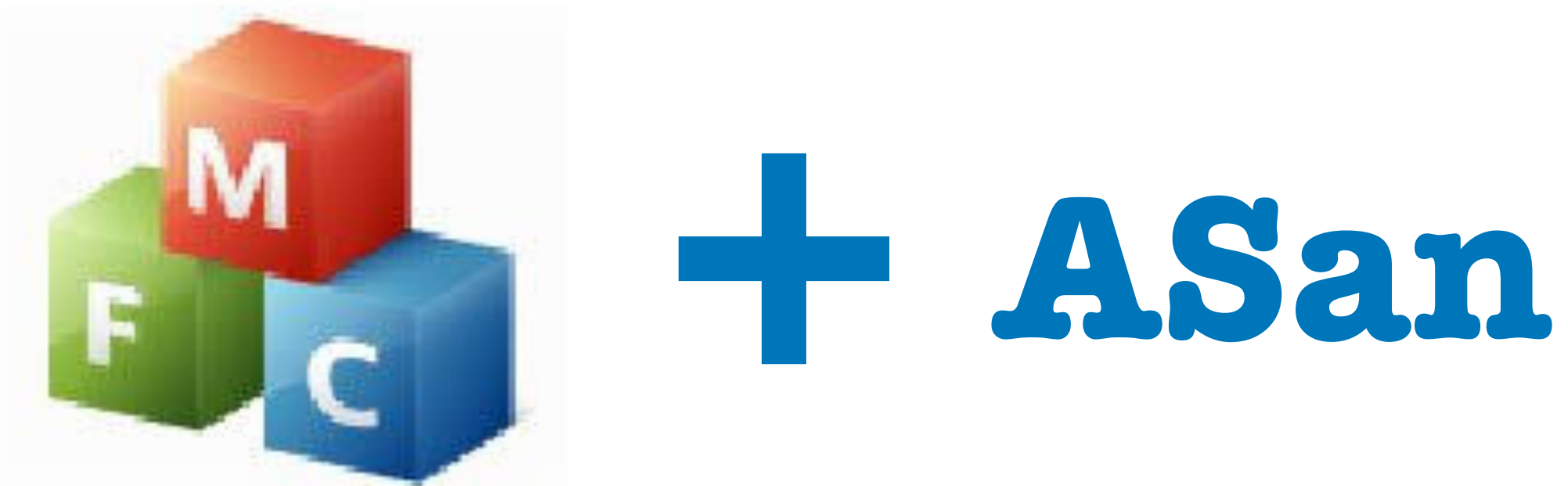
runtime checks and address sanitizer is not currently supported - disabling runtime checks

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



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

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

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

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

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

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



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

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

Workarounds:

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

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\vc\tools\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) ≠ 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 Panel: This panel shows a search bar at the top with the text "Search (Ctrl+E)" and a search icon. Below it, there are navigation arrows and a "Search Depth" dropdown set to "3". A table lists the current stack frame's variables:

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 Panel: This panel shows the sequence of function calls leading to the current state. 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

Process: [24324] TestPlatform.exe | Lifecycle Events | Thread: [24232] Main Thread | Stack Frame: StringUtil::StoreNULLSeparatedStrings

Memory 1

Address: 0x00007FF7E82AC0F4 | Columns: Auto

Address	Hex	ASCII
0x00007FF7E82AC0F4	00 00 00 00 00 00 00 00 00 00 00 00 74 00 6f 00 6b 00 65 00 6e 00 30 00 00 00 74 00 6f 00 6b 00 65 00 6e 00 31 00 00 00t.o.k.e.n.0...t.o.k.e.n.1...
0x00007FF7E82AC11C	74 00 6f 00 6b 00 65 00 6e 00 32 00 00 00 f9 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	t.o.k.e.n.2...ù.....
0x00007FF7E82AC144	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00007FF7E82AC16C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 74 00 65 00 73 00 74 00 31 00 2c 00 74 00 65 00 73 00 74 00t.e.s.t.1.,.t.e.s.t.
0x00007FF7E82AC194	32 00 2c 00 74 00 65 00 73 00 74 00 33 00 00 00 f9 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	2.,.t.e.s.t.3...ù.....
0x00007FF7E82AC1BC	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

StringProcessingTests.cpp | StringProcessing.cpp | atlsimpcoll.h | atlbase.h | SymbolUtil.cpp

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...



Part III

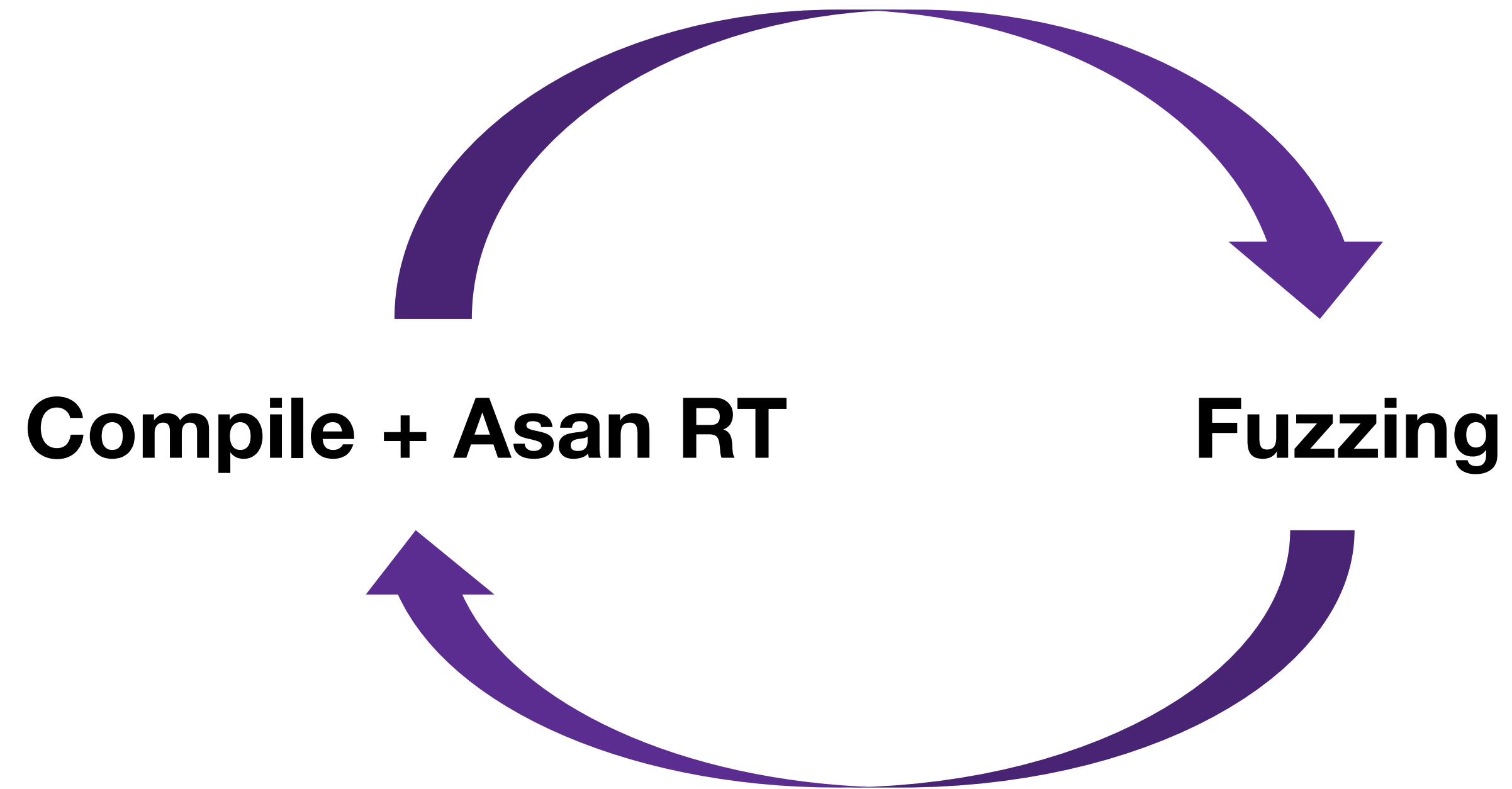
Warm Fuzzy Feelings

Sanitizers + Fuzzing

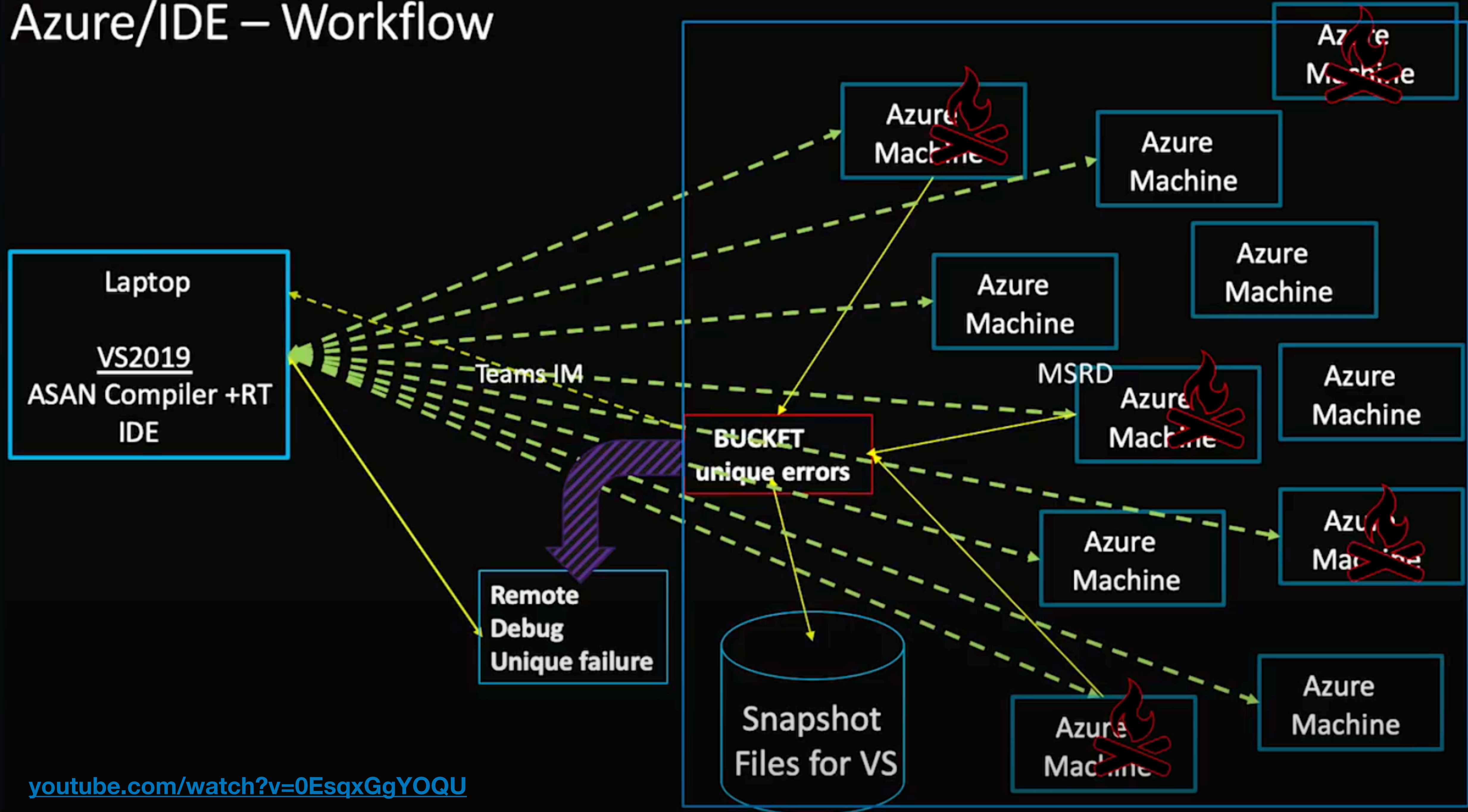


Automatically generate inputs to you program to crash it.

Workflow



Azure/IDE – Workflow



youtube.com/watch?v=0EsqxGgYOQU

Project OneFuzz

September 15, 2020

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

Justin Campbell Principal Security Software Engineering Lead, Microsoft Security

Mike Walker Senior Director, Special Projects Management, Microsoft Security

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

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

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

github.com/microsoft/onefuzz

Project OneFuzz

CI/CD



New unique crashes create notifications:

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



Azure DevOps Pipeline



GitHub Actions

github.com/microsoft/onefuzz-samples

{ ASan + Fuzzing } => Azure

Microsoft Security Risk Detection

Fuzzing Jobs

Fuzzing Jobs

Create Job

Id	Name	OS Image	Created	Status	Results	Actions
8ee12290	Package CppConFuzzTargetVcAsan by jradigan from JRADIGAN-DELLLT	Windows Server 2019 Datacenter x64	9/18/19 1:44 PM	Fuzzing (Day 1 of 14) Started on: 9/18/19 2:09 PM	4	[Icons: List, 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: List, 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: List, Delete, Stop, Refresh]
6852ebcc	Package CppConFuzzTargetVcAsan	Windows Server 2019 Datacenter x64	9/13/19 9:11 AM	Stopped	5	[Icons: List, 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: List, Delete, Stop, Refresh]
a3d2b069	Package CppConFuzzTargetVcAsan	Windows Server 2019 Datacenter x64	9/7/19 11:46 PM	Stopped	5	[Icons: List, Delete, Stop, Refresh]

Azure MSRD service

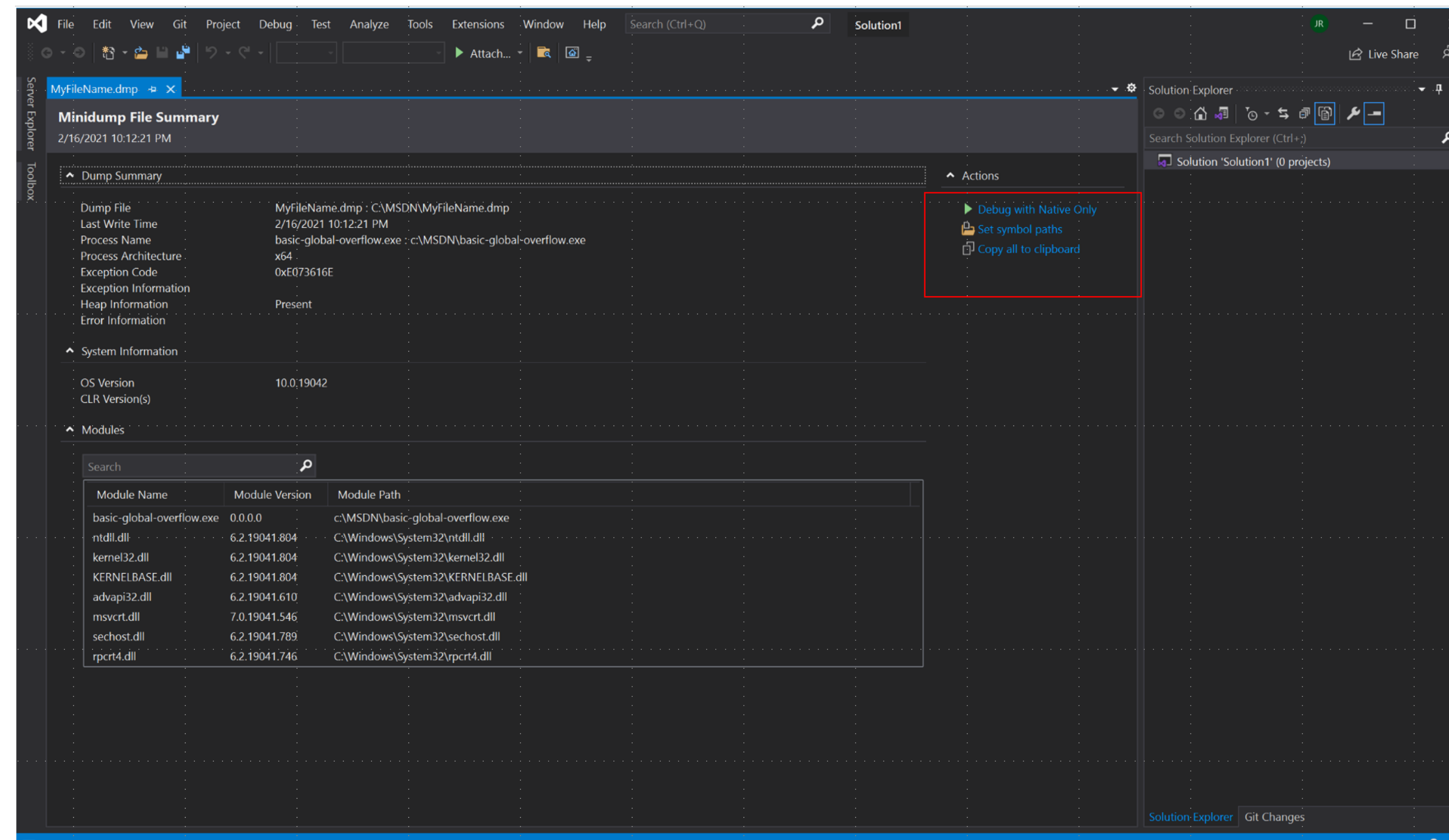
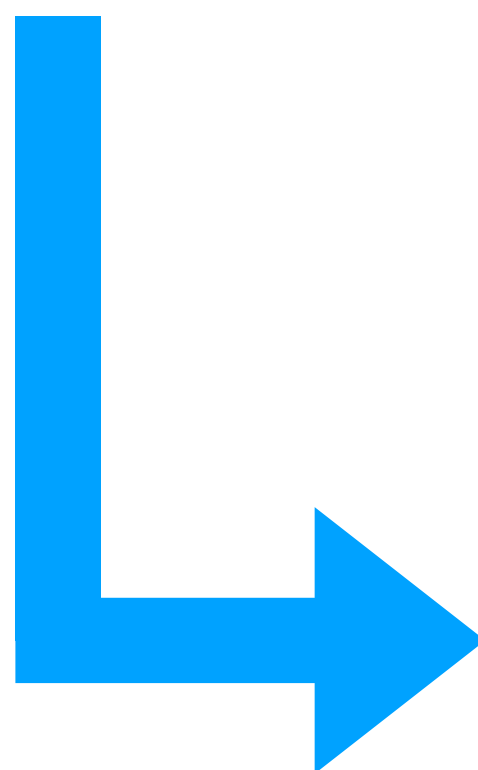
Contact us Privacy & cookies Terms of use Trademarks Third Party Notices © Microsoft 2019

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

Q & A

edu@caphyon.com

Memory Safety: Static vs Dynamic Analysis

Open4Tech

January 19, 2022



@ciura_victor

Victor Ciura
Principal Engineer



CAPHYON