black hat ASIA 2022

Bypass CFG in Chrome

Yunhai Zhang

NSFOCUS

Who am I

Head of NSFOCUS Tianji Lab

Security Researcher

Microsoft Mitigation Bypass Bounty Winner

Pwnie Awards Nominee

What is CFG

Control Flow Guard (CFG) is a highly-optimized platform security feature that was created to combat memory corruption vulnerabilities. By placing tight restrictions on where an application can execute code from, it makes it much harder for exploits to execute arbitrary code through vulnerabilities such as buffer overflows. CFG extends previous exploit mitigation technologies such as /GS, DEP, and ASLR.

- Prevent memory corruption and ransomware attacks.
- Restrict the capabilities of the server to whatever is needed at a particular point in time to reduce attack surface.
- Make it harder to exploit arbitrary code through vulnerabilities such as buffer overflows.

This feature is available in Microsoft Visual Studio 2015, and runs on "CFG-Aware" versions of Windows—the x86 and x64 releases for Desktop and Server of Windows 10 and Windows 8.1 Update (KB3000850).

We strongly encourage developers to enable CFG for their applications. You don't have to enable CFG for every part of your code, as a mixture of CFG enabled and non-CFG enabled code will execute fine. But failing to enable CFG for all code can open gaps in the protection. Furthermore, CFG enabled code works fine on "CFG-Unaware" versions of Windows and is therefore fully compatible with them.

History of CFG

June 2013, first introduced in Windows 8.1 Preview

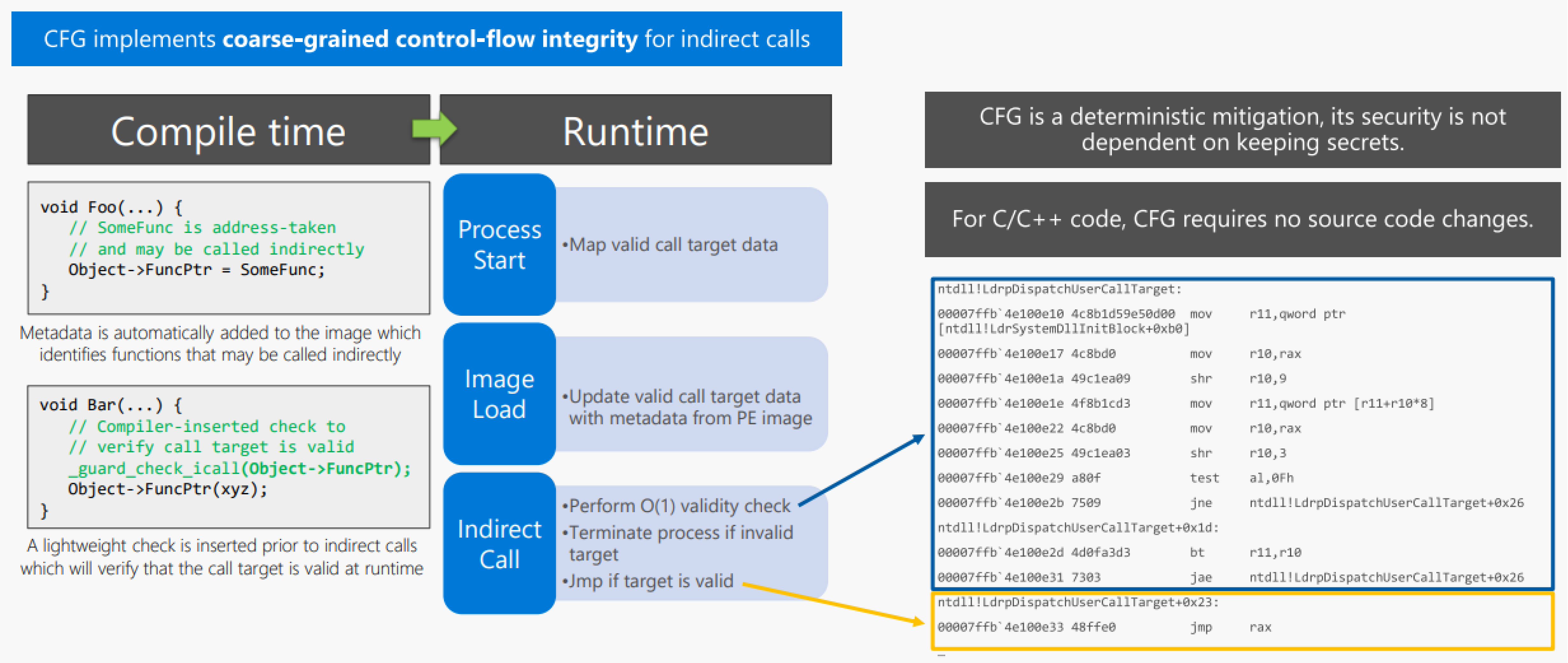
October 2013, disabled in Windows 8.1

July 2015, enabled in Windows 10 version 1507

August 2016, add longjmp protection in Windows 10 version 1607

April 2017, add Export Suppression and Strict Mode in Windows 10 version 1703

How Dose CFG Work



https://github.com/microsoft/MSRC-Security-Research/blob/master/presentations/2018_02_OffensiveCon/The%20Evolution%20of%20CFI%20Attacks%20and%20Defenses.pdf

Previous CFG Bypass

case 19158: AtlThunkPool RWX Page

case 20267: Predict JIT Page

case 31464: JIT Heap Management Cheating

case 32706: Wrapper Functions

case 33039: Make AtlThunkPool Great Again

case 36944: Bad Hook

case 38460: Export Suppression Logical Flaw

case 42895: Manipulate CFG Bitmap

Previous CFG Bypass

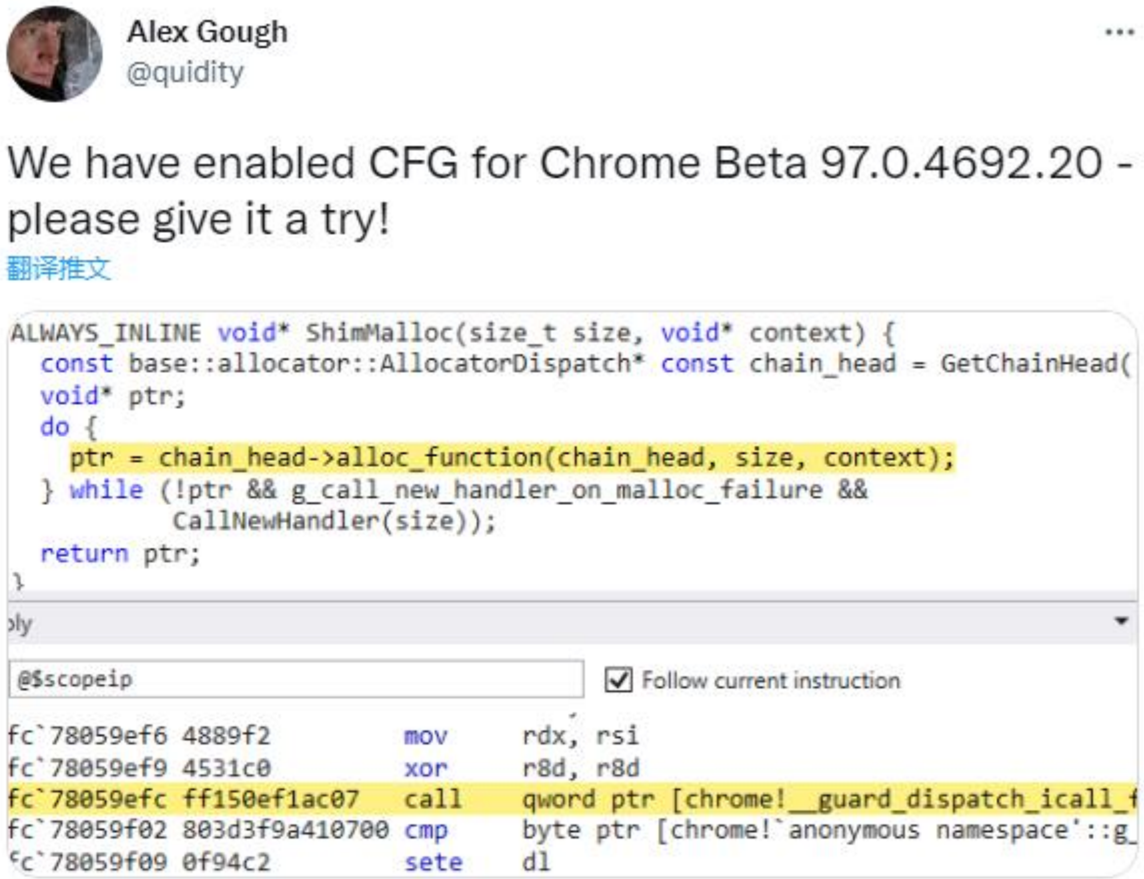
Mitigation	In scope	Out of scope
Control Flow Guard (CFG)	Techniques that make it possible to gain control of the instruction pointer through an indirect call in a process that has enabled CFG.	• Hijacking control flow via return address corruption • Bypasses related to limitations of coarse-grained CFI (e.g. calling functions out of context) • Leveraging non-CFG images • Bypasses that rely on modifying or corrupting read-only memory • Bypasses that rely on CONTEXT record corruption • Bypasses that rely on race conditions or exception handling • Bypasses that rely on thread suspension • Instances of missing CFG instrumentation prior to an indirect call • Code replacement attacks

Previous CFG Bypass

Mitigation	In scope	Out of scope
Control Flow Guard (CFG)	Techniques that make it possible to gain control of the instruction pointer through an indirect call in a process that has enabled CFG.	• Mitigated by CET • Bypasses related to limitations of coarse-grained CFI (e.g. calling functions out of context) • Mitigated by Strict Mode • Bypasses that rely on modifying or corrupting read-only memory • Bypasses that rely on CONTEXT record corruption • Bypasses that rely on race conditions or exception handling • Bypasses that rely on thread suspension • Instances of missing CFG instrumentation prior to an indirect call • Code replacement attacks

How about Chrome

CFG is enabled in Chrome Beta 97.0.4692.20



Alex Gough
@quidity

We have enabled CFG for Chrome Beta 97.0.4692.20 - please give it a try!
翻译推文

```
ALWAYS_INLINE void* ShimMalloc(size_t size, void* context) {
  const base::allocator::AllocatorDispatch* const chain_head = GetChainHead(
  void* ptr;
  do {
    ptr = chain_head->alloc_function(chain_head, size, context);
  } while (!ptr && g_call_new_handler_on_malloc_failure &&
           CallNewHandler(size));
  return ptr;
}
```

ply

@$scopeip   Follow current instruction

```
fc`78059ef6 4889f2         mov    rdx, rsi
fc`78059ef9 4531c0         xor    r8d, r8d
fc`78059efc ff150ef1ac07   call   qword ptr [chrome!__guard_dispatch_icall_f
fc`78059f02 803d3f9a410700 cmp    byte ptr [chrome!`anonymous namespace'::g_
fc`78059f09 0f94c2         sete   dl
```

https://twitter.com/quidity/status/1461427137101979650

How about Chrome

Export Suppression is not enabled

```
(*((ntdll!_EPROCESS *)0xffffc00320909080)).MitigationFlagsValues [Type: <unnamed-tag>]
    [+0x000 ( 0: 0)] ControlFlowGuardEnabled : 0x1 [Type: unsigned long]
    [+0x000 ( 1: 1)] ControlFlowGuardExportSuppressionEnabled : 0x0 [Type: unsigned long]
    [+0x000 ( 2: 2)] ControlFlowGuardStrict : 0x0 [Type: unsigned long]
    [+0x000 ( 3: 3)] DisallowStrippedImages : 0x0 [Type: unsigned long]
    [+0x000 ( 4: 4)] ForceRelocateImages : 0x0 [Type: unsigned long]
    [+0x000 ( 5: 5)] HighEntropyASLREnabled : 0x1 [Type: unsigned long]
    [+0x000 ( 6: 6)] StackRandomizationDisabled : 0x0 [Type: unsigned long]
    [+0x000 ( 7: 7)] ExtensionPointDisable : 0x1 [Type: unsigned long]
    [+0x000 ( 8: 8)] DisableDynamicCode : 0x0 [Type: unsigned long]
    [+0x000 ( 9: 9)] DisableDynamicCodeAllowOptOut : 0x0 [Type: unsigned long]
    [+0x000 (10:10)] DisableDynamicCodeAllowRemoteDowngrade : 0x0 [Type: unsigned long]
    [+0x000 (11:11)] AuditDisableDynamicCode : 0x0 [Type: unsigned long]
    [+0x000 (12:12)] DisallowWin32kSystemCalls : 0x1 [Type: unsigned long]
    [+0x000 (13:13)] AuditDisallowWin32kSystemCalls : 0x1 [Type: unsigned long]
    [+0x000 (14:14)] EnableFilteredWin32kAPIs : 0x0 [Type: unsigned long]
    [+0x000 (15:15)] AuditFilteredWin32kAPIs : 0x0 [Type: unsigned long]
    [+0x000 (16:16)] DisableNonSystemFonts : 0x1 [Type: unsigned long]
    [+0x000 (17:17)] AuditNonSystemFontLoading : 0x0 [Type: unsigned long]
    [+0x000 (18:18)] PreferSystem32Images : 0x0 [Type: unsigned long]
    [+0x000 (19:19)] ProhibitRemoteImageMap : 0x1 [Type: unsigned long]
    [+0x000 (20:20)] AuditProhibitRemoteImageMap : 0x0 [Type: unsigned long]
    [+0x000 (21:21)] ProhibitLowILImageMap : 0x1 [Type: unsigned long]
    [+0x000 (22:22)] AuditProhibitLowILImageMap : 0x0 [Type: unsigned long]
    [+0x000 (23:23)] SignatureMitigationOptIn : 0x1 [Type: unsigned long]
    [+0x000 (24:24)] AuditBlockNonMicrosoftBinaries : 0x0 [Type: unsigned long]
    [+0x000 (25:25)] AuditBlockNonMicrosoftBinariesAllowStore : 0x0 [Type: unsigned long]
    [+0x000 (26:26)] LoaderIntegrityContinuityEnabled : 0x0 [Type: unsigned long]
    [+0x000 (27:27)] AuditLoaderIntegrityContinuity : 0x0 [Type: unsigned long]
    [+0x000 (28:28)] EnableModuleTamperingProtection : 0x0 [Type: unsigned long]
    [+0x000 (29:29)] EnableModuleTamperingProtectionNoInherit : 0x0 [Type: unsigned long]
    [+0x000 (30:30)] RestrictIndirectBranchPrediction : 0x1 [Type: unsigned long]
    [+0x000 (31:31)] IsolateSecurityDomain : 0x0 [Type: unsigned long]
```

How about Chrome

Strict Mode is not enable

```
(*((ntdll!_EPROCESS *)0xffffc00320909080)).MitigationFlagsValues [Type: <unnamed-tag>]
    [+0x000 ( 0: 0)] ControlFlowGuardEnabled : 0x1 [Type: unsigned long]
    [+0x000 ( 1: 1)] ControlFlowGuardExportSuppressionEnabled : 0x0 [Type: unsigned long]
    [+0x000 ( 2: 2)] ControlFlowGuardStrict : 0x0 [Type: unsigned long]
    [+0x000 ( 3: 3)] DisallowStrippedImages : 0x0 [Type: unsigned long]
    [+0x000 ( 4: 4)] ForceRelocateImages : 0x0 [Type: unsigned long]
    [+0x000 ( 5: 5)] HighEntropyASLREnabled : 0x1 [Type: unsigned long]
    [+0x000 ( 6: 6)] StackRandomizationDisabled : 0x0 [Type: unsigned long]
    [+0x000 ( 7: 7)] ExtensionPointDisable : 0x1 [Type: unsigned long]
    [+0x000 ( 8: 8)] DisableDynamicCode : 0x0 [Type: unsigned long]
    [+0x000 ( 9: 9)] DisableDynamicCodeAllowOptOut : 0x0 [Type: unsigned long]
    [+0x000 (10:10)] DisableDynamicCodeAllowRemoteDowngrade : 0x0 [Type: unsigned long]
    [+0x000 (11:11)] AuditDisableDynamicCode : 0x0 [Type: unsigned long]
    [+0x000 (12:12)] DisallowWin32kSystemCalls : 0x1 [Type: unsigned long]
    [+0x000 (13:13)] AuditDisallowWin32kSystemCalls : 0x1 [Type: unsigned long]
    [+0x000 (14:14)] EnableFilteredWin32kAPIs : 0x0 [Type: unsigned long]
    [+0x000 (15:15)] AuditFilteredWin32kAPIs : 0x0 [Type: unsigned long]
    [+0x000 (16:16)] DisableNonSystemFonts : 0x1 [Type: unsigned long]
    [+0x000 (17:17)] AuditNonSystemFontLoading : 0x0 [Type: unsigned long]
    [+0x000 (18:18)] PreferSystem32Images : 0x0 [Type: unsigned long]
    [+0x000 (19:19)] ProhibitRemoteImageMap : 0x1 [Type: unsigned long]
    [+0x000 (20:20)] AuditProhibitRemoteImageMap : 0x0 [Type: unsigned long]
    [+0x000 (21:21)] ProhibitLowILImageMap : 0x1 [Type: unsigned long]
    [+0x000 (22:22)] AuditProhibitLowILImageMap : 0x0 [Type: unsigned long]
    [+0x000 (23:23)] SignatureMitigationOptIn : 0x1 [Type: unsigned long]
    [+0x000 (24:24)] AuditBlockNonMicrosoftBinaries : 0x0 [Type: unsigned long]
    [+0x000 (25:25)] AuditBlockNonMicrosoftBinariesAllowStore : 0x0 [Type: unsigned long]
    [+0x000 (26:26)] LoaderIntegrityContinuityEnabled : 0x0 [Type: unsigned long]
    [+0x000 (27:27)] AuditLoaderIntegrityContinuity : 0x0 [Type: unsigned long]
    [+0x000 (28:28)] EnableModuleTamperingProtection : 0x0 [Type: unsigned long]
    [+0x000 (29:29)] EnableModuleTamperingProtectionNoInherit : 0x0 [Type: unsigned long]
    [+0x000 (30:30)] RestrictIndirectBranchPrediction : 0x1 [Type: unsigned long]
    [+0x000 (31:31)] IsolateSecuritySomain : 0x0 [Type: unsigned long]
```

Bypass CFG in Chrome

Issue 1: RWX memory still exists

WebAssembly.Instance

```
0:008> dq 5df30829a768 + 60 l1
00005df3`0829a7c8  00007bcb`c5d21000
0:008> !address 00007bcb`c5d21000

Usage:                  <unknown>
Base Address:           00007bcb`c5d21000
End Address:            00007bcb`c5d22000
Region Size:            00000000`00001000 (    4.000 kB)
State:                  00001000          MEM_COMMIT
Protect:                00000040          PAGE_EXECUTE_READWRITE
Type:                   00020000          MEM_PRIVATE
Allocation Base:        00007bcb`c5d20000
Allocation Protect:     00000001          PAGE_NOACCESS
```

Bypass CFG in Chrome

Issue 2: Export Suppression is not enabled

- Most export functions can be called out of context
- Sensitive API blocklist is prone to miss wrapper functions

```
__int64 __fastcall RtlpLoadUmsDebugRegisterState(__int64 a1)
{
  char v2[1232]; // [rsp+20h] [rbp-4E8h] BYREF

  if ( !a1 )
    return 0xC000000Di64;
  RtlpCopyLegacyContext(a1, v2, 1048592, a1 + 16);
  return ZwContinue(v2, 0i64);
}
```

Bypass CFG in Chrome

Issue 3: Strict Mode is not enabled

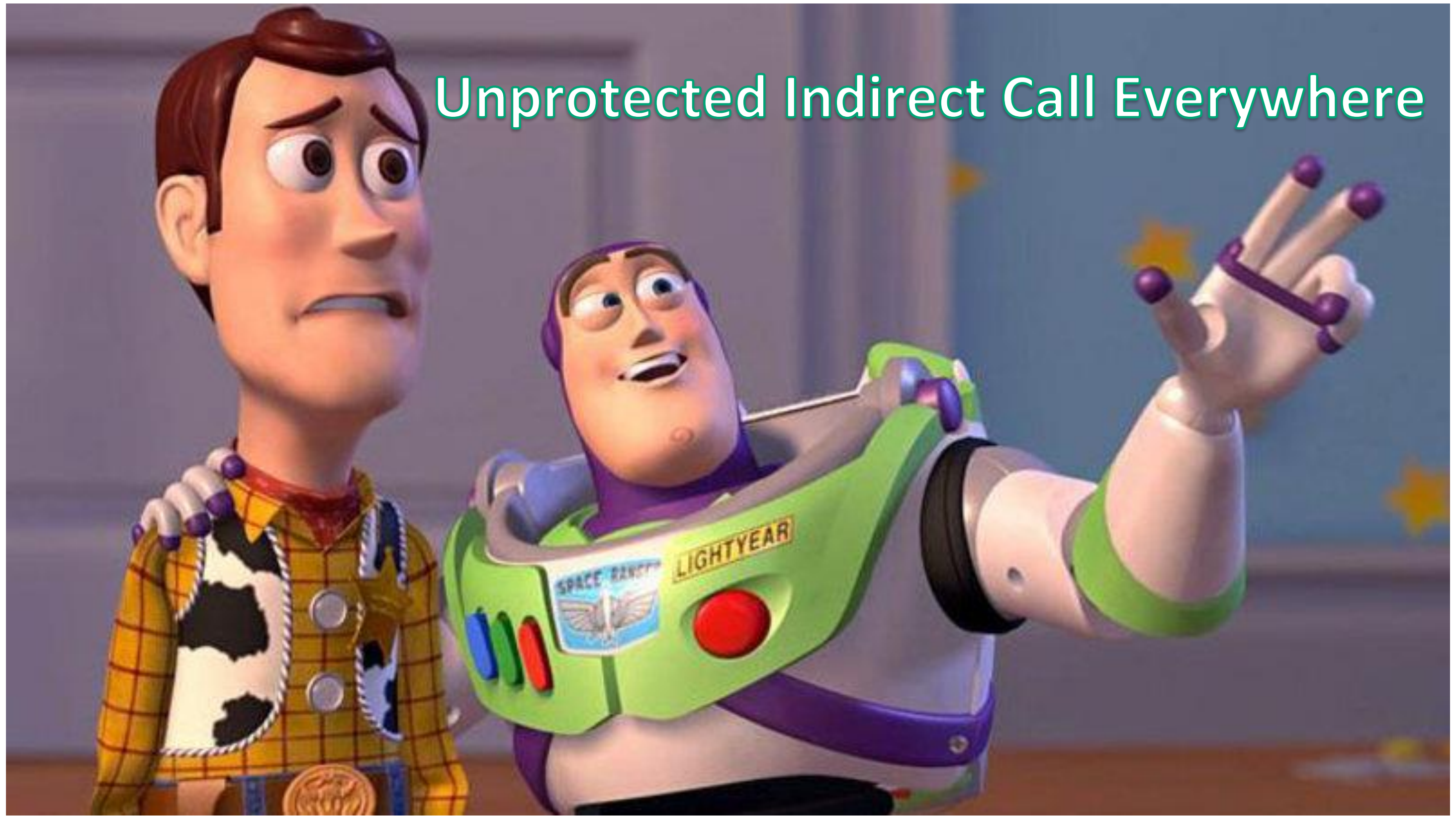
- All address of non-CFG image are valid indirect call target
- There are still dozens of in-box non-CFG DLLs in the latest version of Windows
- The renderer process run in Untrusted IL make it hard to load such a DLL

```
0:012> dq poi(ntdll!LdrSystemDllInitBlock+0xb8) + 00007ff9`75680000 / 40
00007ff5`a934a000  ffffffff`ffffffff ffffffff`ffffffff
00007ff5`a934a010  ffffffff`ffffffff ffffffff`ffffffff
00007ff5`a934a020  ffffffff`ffffffff ffffffff`ffffffff
00007ff5`a934a030  ffffffff`ffffffff ffffffff`ffffffff
00007ff5`a934a040  ffffffff`ffffffff ffffffff`ffffffff
00007ff5`a934a050  ffffffff`ffffffff ffffffff`ffffffff
00007ff5`a934a060  ffffffff`ffffffff ffffffff`ffffffff
00007ff5`a934a070  ffffffff`ffffffff ffffffff`ffffffff
```

Bypass CFG in Chrome

Issue 4: JIT Compiler is CFG unenlightened

Missing CFG instrumentation in JIT generated code



Bypass CFG in Chrome

Issue 4: JIT Compiler is CFG unenlightened

PAGE_TARGETS_INVALID is not used when allocate executable memory

PAGE_TARGETS_INVALID 0x40000000	Sets all locations in the pages as invalid targets for CFG. Used along with any execute page protection like **PAGE_EXECUTE**, **PAGE_EXECUTE_READ**, **PAGE_EXECUTE_READWRITE** and **PAGE_EXECUTE_WRITECOPY**. Any indirect call to locations in those pages will fail CFG checks and the process will be terminated. The default behavior for executable pages allocated is to be marked valid call targets for CFG. This flag is not supported by the VirtualProtect or CreateFileMapping functions.

```
0:015> dq  poi(ntdll!LdrSystemDllInitBlock+0xb8) + 00005deb`00084000 / 40
00007f6d`182c2100  ffffffff`ffffffff ffffffff`ffffffff
00007f6d`182c2110  ffffffff`ffffffff ffffffff`ffffffff
00007f6d`182c2120  ffffffff`ffffffff ffffffff`ffffffff
00007f6d`182c2130  ffffffff`ffffffff ffffffff`ffffffff
00007f6d`182c2140  ffffffff`ffffffff ffffffff`ffffffff
00007f6d`182c2150  ffffffff`ffffffff ffffffff`ffffffff
00007f6d`182c2160  ffffffff`ffffffff ffffffff`ffffffff
00007f6d`182c2170  ffffffff`ffffffff ffffffff`ffffffff
```

Bypass CFG in Chrome

Issue 5: Memory Management can be deceived

Plant evil code into executable memory

```
v8::internal::Factory::CodeBuilder::TryBuild
    v8::internal::Factory::CodeBuilder::BuildInternal
        v8::internal::Factory::CodeBuilder::AllocateCode => The whole code space becomes RWX
        v8::internal::Code::CopyFromNoFlush
        v8::internal::Code::clear_padding    => Only clear a small padding block
        v8::internal::Heap::ProtectUnprotectedMemoryChunks
            v8::internal::MemoryChunk::SetDefaultCodePermissions    => The whole code space becomes R-X
```

Bypass CFG in Chrome

Issue 5: Memory Management can be deceived

Make read-only memory writeable

```
v8::internal::wasm::NativeModule::~NativeModule
    v8::internal::wasm::WasmCodeAllocator::~WasmCodeAllocator
        v8::internal::wasm::WasmCodeManager::FreeNativeModule
            v8::internal::win64_unwindinfo::UnregisterNonABICompliantCodeRange
                VirtualProtect    => The code range becomes RW-
```

Bypass CFG in Chrome

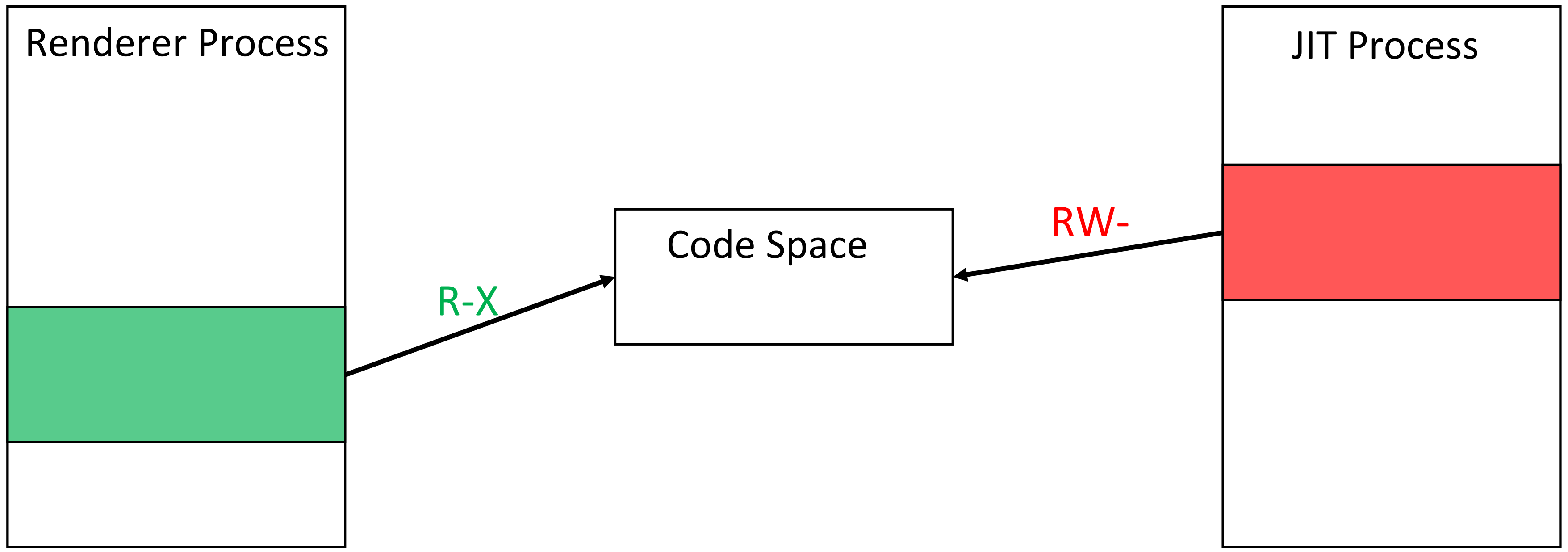
Issue 6: System Weakness

- Writeable critical data structs
- TOCTOU
- Thread suspension

How to Improve

Optimize WebAssembly to eliminate RWX memory

- At least eliminate long-standing RWX memory
- The best practice is to implement OOP code generation



How to Improve

Enable all CFG enhancement

- CFG Export Suppression
- CFG Strict

How to Improve

- Make JIT Compiler CFG enlightened
  - Add CFG instrumentation before indirect call or jump
  - Use PAGE_TARGETS_INVALID when allocating executable memory
  - Use SetProcessValidCallTargets to explicit validate entry points which may be called indirectly
  - Use SetProcessValidCallTargets to invalidate those entry points when releasing them

How to Improve

Switch to the more efficient XFG

Improving Control Flow Integrity

XFG design: basics

Assign a type signature-based tag to each address-taken function

For C-style functions, could be:
hash(type(return_value), type(arg1), type(arg2), ...)

For C++ virtual methods, could be:
***hash**(method_name, type(retval), highest_parent_with_method(type(this), method_name), type(arg1), type(arg2), ...)*

Embed that tag immediately before each function so it can be accessed through function pointer

Add tag check to call-sites: fast fail if we run into a tag mismatch

CFG instrumentation: Call Site

```
mov   rax, [rsi+0x98]                ; load target address
call  [__guard_dispatch_icall_fptr]
```

Target

```
.align 0x10
function:
       push rbp
       push rbx
       push rsi
       ...
```

xFG instrumentation : Call Site

```
mov   rax, [rsi+0x98]                    ; load target address
mov   r10, 0xdeadbeefdeadbeef            ; load function tag
call  [__guard_dispatch_icall_fptr_xfg]  ; will check tag
```

Target

```
.align 0x10
dq 0xcccccccccccccccc ; just alignment
dq 0xdeadbeefdeadbeef ; function tag
function:
       push rbp
       push rbx
       push rsi
```

https://github.com/dwizzzle/Presentations/blob/master/Bluehat%20Shanghai%20-%20Advancing%20Windows%20Security.pdf

Enhanced Security Mode of Edge

Browse more safely with Microsoft Edge

Article • 02/18/2022 • 4 minutes to read • 2 contributors

This article describes how Microsoft Edge provides enhanced security on the web.

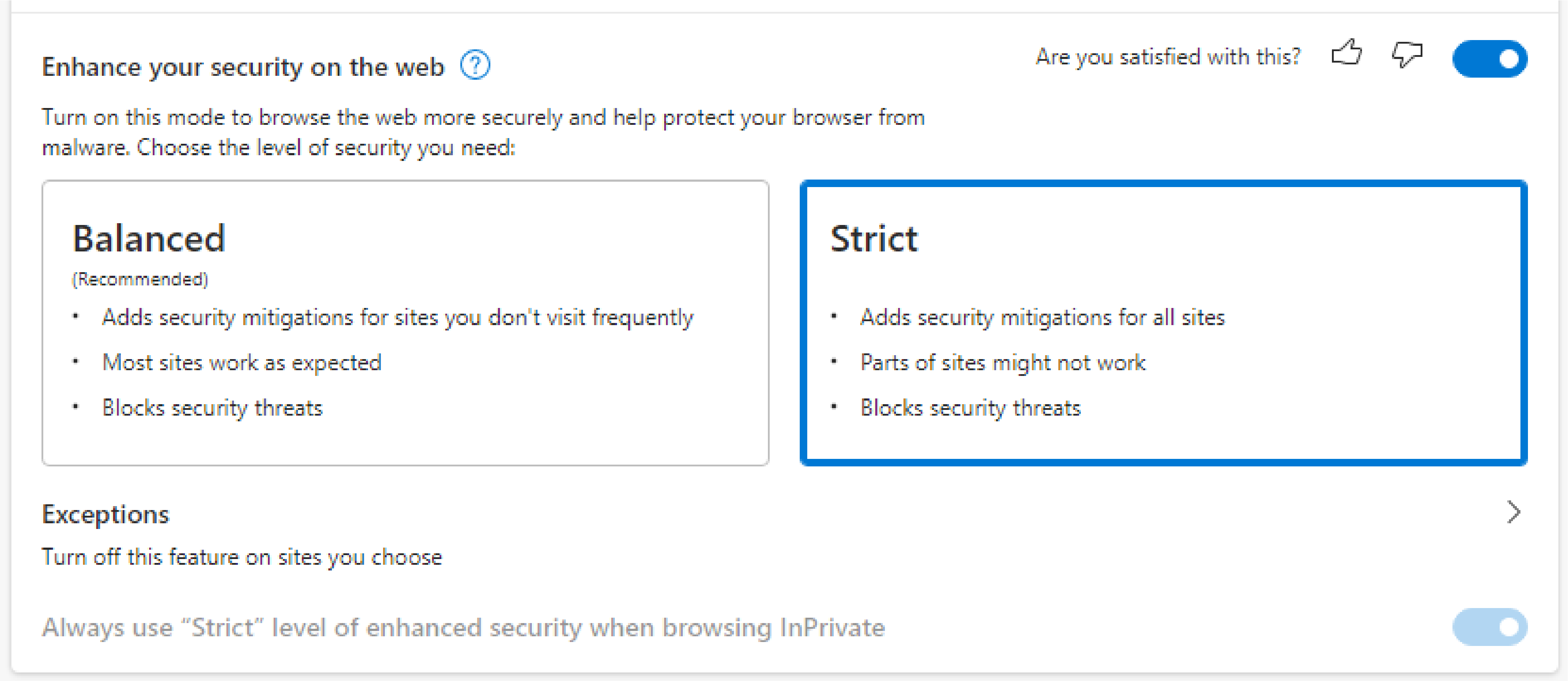
> Note
>
> This article applies to Microsoft Edge version 98 or later.

Overview

Microsoft Edge is adding enhanced security protections to provide an additional layer of protection when browsing the web and visiting unfamiliar sites. The web platform is designed to give you a rich browsing experience using powerful technologies like JavaScript. On the other hand, that power can translate to more exposure when you visit a malicious site. With enhanced security, Microsoft Edge helps reduce the risk of an attack by automatically applying more conservative security settings on unfamiliar sites and adapts over time as you continue to browse.

https://docs.microsoft.com/en-us/DeployEdge/microsoft-edge-security-browse-safer

Enhanced Security Mode of Edge



Enhanced Security Mode of Edge

What this mode do?

- Disabling just-in-time (JIT) JavaScript compilation
- Disabling WebAssembly (WASM) currently
- Enabling Hardware-enforced Stack Protection (CET)
- Enabling Arbitrary Code Guard (ACG)

Enhanced Security Mode of Edge

What this mode means?

- There are no more RWX memory at all
- There are no JIT Compiler at all
- There are no extra executable memory at all

Enhanced Security Mode of Edge

Can we still bypass CFG in this mode?

Pid 6648 - WinDbg:10.0.19041.1 AMD64

File Edit View Debug Window Help

Command

```
ModLoad: 00007ffa`cfe30000 00007ffa`cfe58000   C:\Windows\SYSTEM32\ncrypt.dll
ModLoad: 00007ffa`ba3e0000 00007ffa`ba47b000   C:\Windows\SYSTEM32\WINSPOOL.DRV
ModLoad: 00007ffa`cc890000 00007ffa`cc8ae000   C:\Windows\SYSTEM32\dhcpcsvc.DLL
ModLoad: 00007ffa`cf850000 00007ffa`cf892000   C:\Windows\SYSTEM32\SSPICLI.DLL
ModLoad: 00007ffa`cfd70000 00007ffa`cfd82000   C:\Windows\System32\MSASN1.dll
ModLoad: 00007ffa`cfdf0000 00007ffa`cfe27000   C:\Windows\SYSTEM32\NTASN1.dll
(19f8.a04): Break instruction exception - code 80000003 (first chance)
ntdll!DbgBreakPoint:
00007ffa`d3127360 cc              int     3
0:012> g
(19f8.eec): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
00000000`babeface ??              ???
0:015> r
rax=00000000babeface rbx=000000d90e3fddc0 rcx=00002ff400279c50
rdx=000000d90e3fdc20 rsi=000057590000000a rdi=00000000babeface
rip=00000000babeface rsp=000000d90e3fdb38 rbp=00002ff400b47ff0
 r8=000000d90e3fddc0  r9=00000000babeface r10=0000000008909f00
r11=0000000000000000 r12=000000d90e3fdd90 r13=000000d90e3fddc0
r14=000000d90e3fdc20 r15=000000d90e3fe9d0
iopl=0         nv up ei pl nz na po nc
cs=0033  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010206
00000000`babeface ??              ???
0:015> k 10
 # Child-SP          RetAddr           Call Site
00 000000d9`0e3fdb38 00007ffa`9b818bae 0xbabeface
01 000000d9`0e3fdb40 00007ffa`9a3867d8 msedge!v8::internal::JSArray::SetLength+0x7e
02 000000d9`0e3fdba0 00007ffa`9be34b3a msedge!v8::internal::Accessors::ArrayLengthSetter+0xd8
03 000000d9`0e3fdc90 00007ffa`9b7a062d msedge!v8::internal::Object::SetProperty+0xc4a
04 000000d9`0e3fe4a0 00007ffa`9a352a63 msedge!v8::internal::StoreIC::Store+0x2fd
05 000000d9`0e3fe830 00007ffa`9c2e2bbc msedge!v8::internal::Runtime_StoreIC_Miss+0x153
06 000000d9`0e3fe980 00007ffa`9c37a9c8 msedge!Builtins_CEntry_Return1_DontSaveFPRegs_ArgvOnStack_NoBuiltinExit+0x3c
07 000000d9`0e3fe9d0 00007ffa`9c267522 msedge!Builtins_StaNamedPropertyHandler+0x88
08 000000d9`0e3fea28 00007ffa`9c26555c msedge!Builtins_InterpreterEntryTrampoline+0xe2
09 000000d9`0e3fea78 00007ffa`9c26515b msedge!Builtins_JSEntryTrampoline+0x5c
0a 000000d9`0e3feaa8 00007ffa`9bac1f45 msedge!Builtins_JSEntry+0xdb
0b 000000d9`0e3febc0 00007ffa`9c76dc3e msedge!v8::internal::`anonymous namespace'::Invoke+0xc55
0c 000000d9`0e3feda0 00007ffa`9a364a97 msedge!v8::internal::Execution::CallScript+0xae
0d 000000d9`0e3fee40 00007ffa`9a655bdd msedge!v8::Script::Run+0x557
0e 000000d9`0e3ff090 00007ffa`9c697d60 msedge!blink::V8ScriptRunner::CompileAndRunScript+0x5ed
0f 000000d9`0e3ff3a0 00007ffa`9a2e276d msedge!blink::ClassicScript::RunScriptOnWorkerOrWorklet+0x90
0:015> ub 00007ffa`9b818bae
msedge!v8::internal::JSArray::SetLength+0x5d:
00007ffa`9b818b8d 488b0d2c82d309  mov     rcx,qword ptr [msedge!v8::internal::ElementsAccessor::elements_accessors_ (00007ffa`a5550dc0)]
00007ffa`9b818b94 488b0c41        mov     rcx,qword ptr [rcx+rax*2]
00007ffa`9b818b98 488b01          mov     rax,qword ptr [rcx]
00007ffa`9b818b9b 488b4038        mov     rax,qword ptr [rax+38h]
00007ffa`9b818b9f 4c89f2          mov     rdx,r14
00007ffa`9b818ba2 4989d8          mov     r8,rbx
00007ffa`9b818ba5 4189f9          mov     r9d,edi
00007ffa`9b818ba8 ff156254400a    call    qword ptr [msedge!_guard_dispatch_icall_fptr (00007ffa`a5c1e010)]
```

0:015>

Ln 0, Col 0 | Sys 0:<Local> | Proc 000:19f8 | Thrd 015:eec | ASM | OVR | CAPS | NUM

5:02 PM 4/19/2022

Takeaways

Enabling CFG is not just as simple as open compiler switches

JIT Compiler should be CFG enlightened to avoid abusing

Consider switch to the more efficient XFG

Thanks!