The Power of Data-Oriented Attacks: Bypassing Memory Mitigation Using Data-Only Exploitation Technique
Part I

Bing Sun, Chong Xu, Stanley Zhu
About Speaker

• Bing Sun
  • Bing Sun is a senior information security researcher, and leads the IPS security research team of McAfee. He has extensive experiences in operating system kernel layer and information security R&D, with especially deep diving in advanced vulnerability exploitation and detection, Rootkits detection, firmware security, and virtualization technology.

• Chong Xu
  • Chong Xu received his PhD degree from Duke University with networking and security focus. He is currently a director leading McAfee Labs IPS team, which leads the McAfee Labs vulnerability research, malware and APT detection, botnet detection, and feeds security content and advanced detection features to McAfee's network IPS, host IPS, and firewall products, as well as global threat intelligence.

• Stanley Zhu
  • Stanley Zhu is a security researcher at McAfee. One of designers of Bytehero Heuristic Detection Engine, he provided the BDV engine with Virustotal and OPSWAT platform. He has many years of research experience in information security techniques. He is interested in advanced vulnerability exploitation and detection as well as virus, rootkit and reverse engineering. He has spoken at security conferences such as CanSecWest2014, AVAR2012, XCon2010, XCon2015, and XCon2016.
Abstract

• As Control Flow Integrity (CFI) enforcement solutions are widely adopted by major applications, traditional memory vulnerability exploitation techniques aiming to hijack the control flow have become increasingly difficult. For example, Microsoft’s Control Flow Guard (CFG) is an effective CFI solution against traditional memory exploits. However, due to the CFG implementation limitations, we have seen new exploitation techniques such as using the out-of-context function call to bypass CFG. We believe eventually these limitations could all be overcome or improved, and ultimately we expect a fine-grained CFG solution to completely defeat control-flow hijacking. Consequently, attackers have begun to seek alternatives to exploit memory vulnerabilities without diverting the control flow. As a result of this trend, the data-oriented attacks have emerged. As its name suggests, a data-oriented attack focuses on altering or forging the critical data of an application, rather than attempting to alter its control flow. The data-oriented attack may allow the attacker to do some powerful things, such as loading certain unwanted or disabled modules or changing the attributes of certain memory pages. Sometimes this can be achieved by changing only a few bits of data. Today, most successful memory exploits can gain some level of memory read/write primitives during exploitation of memory corruption vulnerability, which makes data-oriented attacks possible. In this talk, we will present some interesting examples that show the power of data-oriented attacks. We then discuss ways to prevent such attacks. We conclude by live demonstrations of CFG/DEP bypass on Windows 10’s Edge using data-only exploitation technique.
Agenda

• The Principle of CFG
• Known CFG Bypass Methods
  • Corrupt function’s return address on stack
  • Transit via unguarded trampoline code
  • Call function out-of-context
• Call Function Out-of-Context
  • Test an address’ readability by calling kernel32!GlobalLock
  • Bypass CFG by calling ntdll!RtlProtectHeap
• Data-only Attack
  • Bypass CFG by abusing WARP Shader JIT
  • Bypass CFG/DEP by loading disabled Silverlight ActiveX Plugin
• Suggestions for Preventing Data-only Attack
• Conclusion
• Acknowledgement
The Principle of CFG

• About CFG (Control Flow Guard)
  • A compiler-aided exploitation mitigation mechanism that prevents exploit from hijacking the control flow.
  • Compiler inserts CFG check before each indirect control transfer instruction (call/jmp), and at runtime the CFG check will validate the call target address against a pre-configured CFG bitmap to determine whether the call target is valid or not. The process will be terminated upon an unexpected call target being identified.
  • The Relative Virtual Address (RVA) of all valid call targets determined at the time of compilation are kept in a Guard CF Function table in PE file. During the PE loading process, the loader will read CF info from guard CF function table and update the CFG bitmap.
  • The read-only CFG bitmap is maintained by the OS, and part of the bitmap is shared by all processes. An even bit in CFG bitmap corresponds to one 16-bytes aligned address, while an odd bit corresponds to 15 non 16-bytes aligned addresses.
  • When the PE file is loaded, __guard_check_icall_fptr will be resolved to point to ntdll!LdrpValidateUserCallTarget. (on x64, __guard_dispatch_icall_fptr -> ntdll!LdrpDispatchUserCallTarget)
The Principle of CFG (Continued)

Compiler inserts a call target check before each indirect function call/jmp

CFG bitmap base

High 55-bit of call target address is used as an index into the bitmap to get a 64-bit bitmap entry

Bit 3 ~ 8 of target address is used as an offset

Test the bit “offset” of that 64-bit bitmap entry. Target address is valid if bit is set, otherwise trigger INT 29h

Non 16-byte aligned, set bit 0 of offset
Known CFG Bypass Methods

• Corrupt function’s return address on stack
  • “Bypassing Control Flow Guard in Windows 10”
  • Mitigation: RFG (Return Flow Guard), Intel’s CET

• Transit via unguarded trampoline code (mostly involving dynamic code, such as JIT)
  • “Use Chakra engine again to bypass CFG”
  • “Chakra JIT CFG Bypass”
  • Mitigation: JIT security improvement (JIT code checksum, remote JIT etc)

• Call function out-of-context
  • “Bypass Control Flow Guard Comprehensively”
  • “Mitigation bounty — 4 techniques to bypass mitigations”
  • Mitigation: Fine-grained CFI (improvement on CFG after WIP build 14986)
The basic idea of calling function out-of-context

- Issue a function call to certain unexpected target via memory indirect call instruction; however from the program’s logic perspective such a call is not supposed to happen from that call site. This is essentially one type of execution control hijacking.

How to make an out-of-context call

- Overwrite an existing function pointer (such as in an object’s vftable) with the target function of out-of-context call.
- The target function needs to be able to pass the CFG check because almost all memory indirect calls have CFG check prior to them.
- The call to the overwritten function can be reliably and repeatedly triggered from the context of scripting language.
- The number and order of arguments to the target function should be (at least partially) controllable to the scripting language.
- It’s preferable to be able to get the target function’s return value in its original form.

Example

- “From read-write anywhere to controllable calls” This is a very good example of calling function out-of-context, controlling all arguments and getting the return value back.
Test Address’ Readability By Calling kernel32!GlobalLock
Call kernel32!GlobalLock Out-of-context

out-of-context call via RPC

address to test is passed in as the 1st argument
kernel32!GlobalLock

kernel32!GlobalLock is valid for CFG (in WIP build 14986 and before), while kernel32!IsBadReadPtr has been marked as sensitive API. Internally GlobalLock calls IsBadReadPtr that is protected by SEH, so it can be used to reliably test an address’ readability.

.text:0000000180015800 ; LPVOID __stdcall GlobalLock(HGLOBAL hMem)
...
.text:000000018001580A       mov    rbx, rcx
.text:000000018001580D       mov    r8d, 8
.text:0000000180015813       test   r8b, cl
.text:0000000180015816       jz loc_1800158C2 // the bit3 of target address must be cleared
...
.text:00000001800158DB       mov    edx, 1 ; ucb
.text:00000001800158E0       mov    rcx, rbx ; lp
.text:00000001800158E3       call   IsBadReadPtr // call IsBadReadPtr
.text:00000001800158E8       test   eax, eax
.text:00000001800158EA       jz    short loc_1800158B4
.text:00000001800158EC       jmp    loc_18002FF71
.text:00000001800158EC GlobalLock   endp
Bypass CFG By Calling ntdll!RtlProtectHeap
Call ntdll!RtlProtectHeap Out-of-context

out-of-context call via
chakra!Js::JavascriptNativeIntArray::DeleteItem

Disassembly

0x14ff:  mov        ecx, eax
0x1500:  push       ebp
0x1501:  mov        ebp, esp
0x1502:  call       415de4395d chakra!__guard_check_icall_fptr (7d3954dc)
0x1507:  mov        ecx, ebx
0x1508:  mov        ecx, [ebp-4]
0x1509:  mov        ecx, [ebp-14h]
0x150a:  call       415de4395d chakra!__guard_check_icall_fptr (7d3954dc)

Command

0:009; r
eax=00f5b08 ebx=12000180 ecx=12000180 edx=00000100 esi=049f3ce8 edi=04415c10
esp=5ce14f2 esp=049f3ce0 ebp=049f3ce0 iopl=0 nv up ei pl nz pe pe cm ip
cs:001b ss:0023 ds:0023 es:0023 fs:003b gs:0000 efl=00000f04
chakra!Js::JavascriptOperators::OP_DeleteElementI+0x102
5ce14f2: ff5f4 call dword ptr [ebp-4]
5ce14fd: 7407  je chakra!Js::JavascriptOperators::OP_DeleteElementI+0x102
5ce14f9: b904000000 mov ecx, 4
5ce14fd: cd29  int 29h
5ce14fe: eb8a jmp chakra!Js::JavascriptOperators::OP_DeleteElementI+0x9c

1st arg: fake heap with the address to unprotect at _HEAP_SEGMENT.BaseAddress
2nd arg: 0 (unprotect), 1 (protect)
As its name suggests, native API ntdll!RtlProtectHeap can be invoked to protect (read-only, 2\textsuperscript{nd} arg = 1) or unprotect (read/write, 2\textsuperscript{nd} arg = 0) a heap. For example, ntdll.dll relies on this function to protect its critical data on LdrpMrdataHeap. It calls ntdll!RtlpProtectHeap internally to do the actual job of protection change.

```assembly
.text:6A275840 ; int __stdcall RtlProtectHeap(PVOID BaseAddress, char)
.text:6A275840    mov    edi, edi
                  ...
.text:6A275899    cmp    dword ptr [esi+8], 0DDEEDDEEh
.text:6A2758A0    mov    edx, edi    ; NewProtect
.text:6A2758A2    mov    ecx, esi    ; BaseAddress
.text:6A2758A4    jz     short loc_6A2758D4
.text:6A2758A6    call   _RtlpProtectHeap@8 ; RtlpProtectHeap(x,x)
                  ...
.text:6A2758BF    pop    esi
.text:6A2758C0    pop    ebp
.text:6A2758C1    retn   8
```
ntdll!RtlpProtectHeap

ntdll!RtlpProtectHeap makes the memory attribute change by calling system service ntdll!ZwProtectVirtualMemory.

.text:6A2758DB ; __stdcall RtlpProtectHeap(x, x)
...
.text:6A27592C    mov  eax, [ebp+var_24]
.text:6A27592F    mov  [ebp+ProtectSize], eax
.text:6A275932    lea  eax, [ebp+OldProtect]
.text:6A275935    push eax ; OldProtect
.text:6A275936    push [ebp+NewProtect]; NewProtect
.text:6A275939    lea  eax, [ebp+ProtectSize]
.text:6A27593C    mov  [ebp+BaseAddress], esi
.text:6A27593F    push eax ; ProtectSize
.text:6A275940    lea  eax, [ebp+BaseAddress]
.text:6A275943    push eax ; BaseAddress
.text:6A275944    push 0FFFFFFFFh ; ProcessHandle
.text:6A275946    call  _ZwProtectVirtualMemory@20 ; ZwProtectVirtualMemory(x,x,x,x,x)
...
Call ntdll!RtlProtectHeap Out-of-context

Internally ntdll!RtlProtectVirtualMemory is called to perform the actual memory attribute change.

The address to unprotect is preset at _HEAP_SEGMENT.BaseAddress of the fake heap.
CFG Improvement in WIP 15048

kernel32!GlobalLock failed to pass CFG check
CFG Improvement in WIP 15048

ntdll!LdrpDispatchUserCallTargetES failed to pass CFG check
Significant improvement was made on CFG after WIP build 14986, many previous valid call targets now are no longer valid (see previous two slides)! Obviously, MS has been working hard to make its CFG implementation more fine-grained to defeat calling function out-of-context. As a consequence, we’ll have to find new way to exploit memory vulnerability, and this is really where data-only attack come into play!

“Microsoft Mitigation Bypass Bounty”

<table>
<thead>
<tr>
<th>Mitigation</th>
<th>In scope</th>
<th>Out of scope</th>
</tr>
</thead>
</table>
| 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 |
Data-only Attack

• The principle of data-only attack
  • Change the program’s default behavior by manipulating the data it depends on without diverting the program’s execution control flow. Change in program’s data may lead to the change of program’s original execution path, therefore achieve some powerful things, such as bypassing certain restriction or protection. Please note that the change in program’s code path caused by the change of data is still within the program’s normal logic, so it won’t have problem when CFI is enforced.

• Example of successful data-oriented attack
  • Vital Point Strike (JavaScript god mode)
  • EMET bypass (Replacing EnableProtectionPtr in CONFIG_STRUCT)

• What kind of data will be targeted
  • Any data that can be leveraged to alter the program’s default behavior will be of interest to the data-oriented attack, such as certain global flag in data section, or field in an object etc.
  • Unprotected .data section, unprotected heap/private data, stack.

• How to perform the data-only attack
  • Most data-oriented attacks require the ability of arbitrary address read/write.
  • Being able to accurately locating the targeted data at runtime is the key to success.
  • The existence of certain data is transient or time-sensitive, so timing may play an important role in these cases.
CFG Bypass: Abuse WARP Shader JIT

• We discovered a new CFG bypass method and reported it to Microsoft in early 2016, then Microsoft fixed the issue in its Jun’16 Patch Tuesday release.
  • “Microsoft’s June Patch Kills Potential CFG Bypass”
  • “JIT Spraying Never Dies - Bypass CFG By Leveraging WARP Shader JIT Spraying” Full slides of our presentation on XCon 2016

• In later research, we found the June fix can be easily bypassed using data-only attack (by simply flipping a global variable). We reported this to Microsoft again, and finally the issue got completely fixed in Oct’16 😊.

• Let’s try reproducing the whole story by doing a comparison on different versions involved.
The 1st Version

CFG was not even added in version 10.0.10586.0 of d3d10warp.dll
A writable global variable is used to keep the current status of CFG in version 10.0.10586.420 of d3d10warp.dll
In version 10.0.10586.672 of d3d10warp.dll, instead of using global variable, a new function was introduced, and it’ll be called each time when the status of CFG needs to be verified.
That’s it? Is data-only attack always as simple as flipping a bit? Of course NOT! Let’s look at another more complicated example and see how pure data manipulation can be used to complete an impossible task!
CFG/DEP Bypass: Load Disabled ActiveX Extension
Edge Stops the Support for ActiveX (Including Silverlight)

- ActiveX (including Silverlight browser plugin) was no longer supported in Edge.

Microsoft Edge and Silverlight

Support for ActiveX has been discontinued in Microsoft Edge, and that includes removing support for Silverlight. The reasons for this have been discussed in previous blogs and include the emergence of viable and secure media solutions based on HTML5 extensions. Microsoft continues to support Silverlight, and Silverlight out-of-browser apps can continue to use it. Silverlight will also continue to be supported in Internet Explorer 11, so sites continue to have Silverlight options in Windows 10. At the same time, we encourage companies that are using Silverlight for media to begin the transition to DASH/MSE/CENC/EME based designs and to follow a single, DRM-interoperable encoding work flow enabled by CENC. This represents the most broadly interoperable solution across browsers, platforms, content and devices going forward.

- Besides ActiveX, BHO, VML and VBScript were also removed in Edge, and such removal significantly reduced the attack surface of Edge.
Force Silverlight to Be Loaded in Edge

• In order to force Silverlight Plugin to be loaded in Edge, two places need to be modified to bypass the restrictions.
  • The corresponding feature entry in feature cache (urlmon!g_pFeatureCache)
    • Two security checks are implemented when an ActiveX object is instantiated: one is edgehtml!COleSite::CreateObjectSecurityChecks, the other is urlmon!IsActiveXExtensionAllowed. Both security checks call a same function urlmon!CoInternetExtensionAllowedForUri to verify if the ActiveX object is safe to load.
    • urlmon!CoInternetExtensionAllowedForUri checks the 14th entry in feature cache to determine whether internet extension is allowed or not, and the feature cache is managed by urlmon.dll.
  • Search path flag for DLL loading (combase!GetLoadLibraryAlteredSearchPathFlag)
    • For Silverlight plugin module (npctrl.dll) to be successfully loaded by Edge, the dwFlags argument of LoadLibraryExW must be set to LOAD_WITH_ALTERED_SEARCH_PATH.
    • Upon COM module loading, a global variable in combase.dll will be used as the 3rd argument in the call to LoadLibraryExW.
The First Security Check:
edgehtml!COleSite::CreateObjectSecurityChecks

<table>
<thead>
<tr>
<th># Child-SP</th>
<th>RetAddr</th>
<th>: Args to Child</th>
<th>: Call Site</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 000000e3<code>764f6d70 00007ff9</code>c671064e : 00000000<code>00000055 000001ff</code>82378220 00000000<code>00000001 0007ff9</code>b5af07c0</td>
<td>urlmon!CoInternetIsFeatureEnabled+0x4d</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01 000000e3<code>764f6db0 00007ff9</code>b576dd02 : 00000000<code>00000055 00000000</code>000000e3<code>764f6ec1 000000e3</code>764f7030 00000000`00000001</td>
<td>urlmon!CoInternetExtensionAllowedForUri+0x5e</td>
<td></td>
<td></td>
</tr>
<tr>
<td>02 000000e3<code>764f6e00 00007ff9</code>b576eb64 : 00000000<code>00000055 000001ff</code>82350c00 00000000<code>00000000 00000000</code>00000000</td>
<td>edgehtml!COleSite::CreateObjectSecurityChecks+0x8a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>03 000000e3<code>764f6f10 00007ff9</code>b588613c : 00000207<code>962a9340 00000207</code>96296580 00000000e3<code>764f7090 00000207</code>962c0000</td>
<td>edgehtml!COleSite::CreateObject+0x154</td>
<td></td>
<td></td>
</tr>
<tr>
<td>04 000000e3<code>764f6f90 00007ff9</code>b576f30b : 00000207<code>962c0000 00000207</code>962c0000 00000000<code>000002eb 000000e3</code>764f9158</td>
<td>edgehtml!COleSite::FinalCreateObject+0x32c</td>
<td></td>
<td></td>
</tr>
<tr>
<td>05 000000e3<code>764f90e0 00007ff9</code>b5a06d2b : 00000000<code>00000000 0000001ff</code>82303cf0 00000000`00000000</td>
<td>edgehtml!COleSite::CreateObject+0x1b3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>06 000000e3<code>764f91d0 00007ff9</code>b5bf2d58 : 00000000<code>000000b8 00000207</code>82502250 00000000`00000000</td>
<td>edgehtml!CObjectElement::Execute+0x2b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>07 000000e3<code>764f9200 00007ff9</code>b5b92814 : 00000000<code>ffffffff 00000000</code>00000000 0000001ff`8239a9e0</td>
<td>edgehtml!CObjectElement10ParseCtx::Execute+0x22b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>08 000000e3<code>764f9290 00007ff9</code>b5b92250 : 000001ff<code>82356000 000001ff</code>82356000 000001ff`82356000</td>
<td>edgehtml!CDoc::ParseHtmlStream+0x258</td>
<td></td>
<td></td>
</tr>
<tr>
<td>09 000000e3<code>764f92d0 00007ff9</code>b5ba22a7 : 000001ff<code>8232a9e0 000001ff</code>82356000 00000000<code>00000800 000001ff</code>8232a9e0</td>
<td>edgehtml!CDoc::Broadcast+0x54</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0a 000000e3<code>764f94b0 00007ff9</code>b5b8b20c : 000001ff<code>82356000 00000000e3</code>764f9589 000000e3`764f9670</td>
<td>edgehtml!CDoc::Exec+0x180</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0b 000000e3<code>764f9500 00007ff9</code>b5b8a5c4 : 00000000<code>00000003 00000000</code>00000008 00000000`00000000</td>
<td>edgehtml!CDoc::PerformSyncParse+0x14b</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The Second Security Check:
urlmon!IsActiveXExtensionAllowed

0:019> kv

# Child-SP RetAddr : Args to Child : Call Site
00 000000e3`764f5a70 00007ff9`c671064e : 00000000`00000000 00007ff9`c672614a 00000000`00000000 00007ff9`b6a11878 : urlmon!CoInternetIsFeatureEnabled+0x4d
01 000000e3`764f5ab0 00007ff9`c671033a : 00000000`00000000 00007ff9`b6a11878 00000000e3`764f5c68 00000000e3`764f5c68 : urlmon!CoInternetExtensionAllowedForUri+0x5e
02 000000e3`764f5b00 00007ff9`c6710551 : 00000000`00000000 00000207`857c10a0 00000000`00000000 00000000`00000000 00007ff9`c6710306 : urlmon!IsActiveXExtensionAllowed+0x26
03 000000e3`764f5b30 00007ff9`c670f059 : 00000207`83113e50 00000000e3`764f5c80 000000207`857c10a0 00000000`00000000 : urlmon!CoGetClassObjectForObjectBinding+0x75
04 000000e3`764f5b80 00007ff9`b5885967 : 000001ff`80347d70 00000000`00000004 00000207`9628f818 000000207`9628f818 : urlmon!CoGetClassObjectFromURLInternal+0x2d9
05 000000e3`764f6d30 00007ff9`b576ef69 : 00000207`00000000 00000207`00000000 000000e3`00000000 00000000`00000000 : edgehtml!ICodeLoad::BindToObject+0x3eb
06 000000e3`764f6ea0 00007ff9`b576ebcb : 00000000`00000055 000001ff`82350c00 00000000`00000000 00000000`00000055 : edgehtml!ICodeLoad::Init+0x2b9
07 000000e3`764f6a10 00007ff9`b588613c : 00000000`00000001 00000207`96296580 00000000e3`764f7090 000000207`962c0000 : edgehtml!CObjectSite::CreateObject+0x1bb
08 000000e3`764f6f90 00007ff9`b576f30b : 00000207`962c0000 00000207`962c0000 00000000`00000000`000002eb 00000000e3`764f9158 : edgehtml!CObjectElement::FinalCreateObject+0x32c
09 000000e3`764f90e0 00007ff9`b5a06d2b : 00000000`00000000 000001ff`82303cf0 00000000`ffffffff 00000000`00000000 : edgehtml!CHtmParseCtx::Execute+0x2b
0a 000000e3`764f91d0 00007ff9`b5bf2d58 : 00000000`000003b8 00000207`82502250 00000000`00000000`00000000 : edgehtml!CHtmParseBase::Execute+0x2b
0b 000000e3`764f9200 00007ff9`b5b92814 : 00000000`ffffffff 00000000`00000000 : edgehtml!CHtmParseBase::Execute+0x2b
...
Both security checks call `urlmon!CoInternetExtensionAllowedForUri` to determine whether the ActiveX object is allowed to be loaded. This function calls `urlmon!CoInternetIsFeatureEnabled` to check the status of 14th feature in feature cache.

```asm
.text:00000001800105F0 ; int __fastcall CoInternetExtensionAllowedForUri(_GUID *rclsid, unsigned int dwExtensionType, IUri *pUri, int fAllowWebBrowserInImmersiveMode)
.text:00000001800105F0 xchg ax, ax
.text:00000001800105F2 nop dword ptr [rax+00000000h]
.text:00000001800105F9 mov r11, rsp ...
.text:0000000180010643 lea edx, [rdi+00000000h] ; dwFlags
.text:0000000180010646 lea ecx, [rdi+00000000h] ; FeatureEntry // ecx = 0x0d, test the 14th feature entry
.text:0000000180010649 call CoInternetIsFeatureEnabled
.text:000000018001064E lea ebx, [rdi+1]
.text:0000000180010651 cmp eax, ebx
.text:0000000180010653 jz loc_180010703...
```
urlmon!CoInternetIsFeatureEnabled tests the specified feature entry and returns whether the corresponding feature is enabled or not.

```plaintext
urlmon!CoInternetIsFeatureEnabled tests the specified feature entry and returns whether the corresponding feature is enabled or not.

_test:000000001800260A0 ; HRESULT __stdcall CoInternetIsFeatureEnabled(INTERNETFEATURELIST FeatureEntry, DWORD dwFlags)
...
_test:000000001800260CA    mov    rdi, cs:?g_pFeatureCache@@3PEAVCFeatureCache@@EA ; // pointer to feature cache
...
_test:000000001800260DA    mov    ecx, esi
_test:000000001800260DC    mov    ebx, 1
_test:000000001800260E1    shl    rbx, cl // compute the entry to test
...
_test:000000001800260E9    test   [rdi+8], rbx // test the corresponding bit, in this case rbx=0x2000
_test:000000001800260ED    mov    ebx, 0
..test:000000001800260F2    setz   bl
_test:000000001800260F5    test   ebx, ebx
..test:000000001800260F7    js     loc_18002619D ; jumptable 000000018002616D default case
```
urlmon!g_pFeatureCache

- The feature cache is not protected for write, and the default feature entry for internet extension is 1, which means ActiveX object is NOT allowed. Changing it to 0 allows the Silverlight to be loaded.
Search Path Flag for DLL Loading:
combase!CClassCache::CDllPathEntry::LoadDll

0:019> kv

<table>
<thead>
<tr>
<th># Child-SP</th>
<th>RetAddr</th>
<th>: Args to Child</th>
<th>: Call Site</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000000e3'764f39d0 00007ff9'd1d237b4</td>
<td>: 003000045<code>00380032 00007ff9</code>d4805136 00390039<code>002d0039 0042002d</code>00450046</td>
<td>: combase!GetLoadLibraryAlteredSearchPathFlag+0x13</td>
</tr>
<tr>
<td>01</td>
<td>0000000e3'764f3a10 00007ff9'd1d235c4</td>
<td>: 0000000e3<code>764f3b60 000000e3</code>764f3ae0 000000e3<code>764f3b40 00000000</code>00000000 : combase!CClassCache::CDllPathEntry::LoadDll+0x48</td>
<td></td>
</tr>
<tr>
<td>02</td>
<td>0000000e3`764f3a70 00007ff9'd1d74e59</td>
<td>: 0000000e3<code>764f3b90 000000e3</code>764f3ae0 000000e3<code>764f3b60 00000000</code>00000000 : combase!CClassCache::CDllPathEntry::Create+0x58</td>
<td></td>
</tr>
<tr>
<td>03</td>
<td>0000000e3`764f3b10 00007ff9'd1d3adb2</td>
<td>: 00000207<code>831df60 00000207</code>831df60 00000000<code>00000000 000000e3</code>764f4290</td>
<td>: combase!CClassCache::CDllPathEntry::CreateDllClassEntry+0xf5</td>
</tr>
<tr>
<td>04</td>
<td>0000000e3`764f3de0 00007ff9'd1d7772d</td>
<td>: 00000000<code>00000000 00007ff9</code>00000000 000000e3<code>00000000 0000001ff</code>80390840</td>
<td>: combase!CClassCache::GetClassObjectActivator+0x7f2</td>
</tr>
<tr>
<td>05</td>
<td>0000000e3`764f41b0 00007ff9'd1d77258</td>
<td>: 0000001ff<code>80390840 000000e3</code>764f4e28 00000000<code>00000000 00000000</code>00000000 : combase!CClassCache::GetClassObject+0x4d</td>
<td></td>
</tr>
<tr>
<td>06</td>
<td>(Inline Function)</td>
<td>: --------- : --------- : ---------</td>
<td>: combase!CCGetClassObject+0x52 (Inline Function @ 00007ff9`d1d77258)</td>
</tr>
<tr>
<td>07</td>
<td>0000000e3`764f4220 00007ff9'd1d7c841</td>
<td>: 00000000<code>00000000 00000000</code>00000000 000000e3`764f4e28</td>
<td>: combase!CCServerContextActivator::CreateInstance+0x178</td>
</tr>
<tr>
<td>08</td>
<td>0000000e3`764f4320 00007ff9'd1d7799a9</td>
<td>: 00000000 00000000 00000000 00000000 : combase!ActivationPropertiesIn::DelegateCreateInstance+0xe1</td>
<td></td>
</tr>
<tr>
<td>09</td>
<td>0000000e3`764f43b0 00007ff9'd1d77eab</td>
<td>: 00000000 00000000 00000000 00000000 : combase!CApartmentActivator::CreateInstance+0xc9</td>
<td></td>
</tr>
<tr>
<td>0a</td>
<td>0000000e3`764f4460 00007ff9'd1f92380</td>
<td>: 00000000<code>00000000 000000e3</code>764f4f50 000000e3`764f4e28</td>
<td>: combase!CProcessActivator::CICCallback+0x7b</td>
</tr>
<tr>
<td>0b</td>
<td>0000000e3`764f44a0 00007ff9'd1f92370</td>
<td>: 000000e3<code>764f4d0 000000e3</code>764f4580</td>
<td>: combase!CProcessActivator::AttemptActivation+0x49</td>
</tr>
<tr>
<td>0c</td>
<td>0000000e3`764f44f0 00007ff9'd1f92370</td>
<td>: 000000e3<code>764f4e00 000000e3</code>764f4f00 000000e3<code>764f4900 000000e3</code>764f4e28</td>
<td>: combase!CProcessActivator::ActivateByContext+0xb7</td>
</tr>
<tr>
<td>0d</td>
<td>0000000e3`764f4580 00007ff9'd1d7c88d</td>
<td>: 00000000 00000000 00000000 00000000 : combase!CProcessActivator::CreateInstance+0x94</td>
<td></td>
</tr>
</tbody>
</table>
Search Path Flag for DLL Loading (Continued)

0e 000000e3`764f45d0 00007ff9`d1d80132 : 000000e3`764f4c28 000000e3`00000000 000000e3`764f4690 000000e3`764f4c28 : combase!ActivationPropertiesIn::DelegateCreateInstance+0x12d
0f 000000e3`764f4660 00007ff9`d1d7c84b : 000000e3`764f4c28 00000000`00000000 ffffff1c`89b0b610 00000000`00000000 : combase!CClientContextActivator::CreateInstance+0x152
10 000000e3`764f4910 00007ff9`d1d538fe : 00000000`00000000 000000e3`764f5430 00000000`00000000 00000000`00000000 : combase!ActivationPropertiesIn::DelegateCreateInstance+0xeb
11 000000e3`764f49a0 00007ff9`d1d52b80 : 00000000`0000002c 00000000`00000008 00000000`00000002 00000000`00000006 : combase!ICoCreateInstanceEx+0xc0e
12 000000e3`764f56f0 00007ff9`d1d5299c : 00000000`00000000 00000000`00000000 00000000`00000000 : combase!ActivationPropertiesIn::DelegateCreateInstance+0xeb
13 (Inline Function) -------- : -------- '-------- '-------- '-------- '-------- '-------- '-------- '-------- '-------- '-------- : combase!ICoCreateInstanceEx+0x97 (Inline Function @ 00007ff9`d1d5299c)
14 000000e3`764f5810 00007ff9`b5fbe300 : 00000000`00000000 00000000`00000000 00000000`00000000 : combase!ActivationPropertiesIn::DelegateCreateInstance+0xeb
15 000000e3`764f58b0 00007ff9`b5de107c : 00000000`00000000 00000000`00000000 00000000`00000000 : edgehtml!CMarkup::_BuildExtensionValidatorsList+0xb4
16 000000e3`764f5920 00007ff9`b5928f5 : 00000000`00000000 00000000`00000000 00000000`00000000 : edgehtml!CMarkup::ValidateExtension+0x54e744
17 000000e3`764f59b0 00007ff9`b597d625 : 00000000`00000000 00000000`00000000 00000000`00000000 : edgehtml!CCodeLoad::OnProgress+0x135
18 000000e3`764f5ac0 00007ff9`b597d625 : 00000000`00000000 00000000`00000000 00000000`00000000 : urlmon!CExtensionValidationProxy::_SendCLSIDToBSC+0x59
19 000000e3`764f5b00 00007ff9`b597d625 : 00000000`00000000 00000000`00000000 00000000`00000000 : urlmon!CExtensionValidationProxy::_SendCLSIDToBSCInBindCtx+0x75
1a 000000e3`764f5b30 00007ff9`b597d625 : 00000000`00000000 00000000`00000000 00000000`00000000 : urlmon!CExtensionValidationProxy::_SendCLSIDToBSCInBindCtx+0x75
1b 000000e3`764f5b80 00007ff9`b5885967 : 00000000`00000000 00000000`00000000 00000000`00000000 : urlmon!CExtensionValidationProxy::SendCLSIDToBSC+0x59
1c 000000e3`764f6d30 00007ff9`b576ef6f : 00000000`00000000 00000000`00000000 00000000`00000000 : edgehtml!CCodeLoad::BindToObject+0x3eb
1d 000000e3`764f6ea0 00007ff9`b576ebcb : 00000000`00000000 00000000`00000000 00000000`00000000 : urlmon!CExtensionValidationProxy::SendCLSIDToBSCInBindCtx+0x75
1e 000000e3`764f6f10 00007ff9`b588613c : 00000000`00000000 00000000`00000000 00000000`00000000 : urlmon!CExtensionValidationProxy::SendCLSIDToBSCInBindCtx+0x75
...
Before calling LoadLibraryExW, combase!CClassCache::CDllPathEntry::LoadDll calls GetLoadLibraryAlteredSearchPathFlag to obtain the search path flag for DLL loading.

```assembly
.text:000000018004376C ; HRESULT __fastcall CClassCache::CDllPathEntry::LoadDll(DLL_INSTANTIATION_PROPERTIES *dip, HRESULT (__fastcall **pfGetClassObject)(_GUID *, _GUID *, void **), HRESULT (__fastcall **pfGetActivationFactory)(HSTRING __ *, IActivationFactory **), HRESULT (__fastcall **pfDllCanUnload)(), HINSTANCE __ **hDll)

.text:000000018004376C  xchg  ax, ax
.text:000000018004376E  nop  dword ptr [rax+00000000h]
.text:0000000180043775  mov  [rsp+arg_0], rbx

.text:00000001800437AF                 call    ?GetLoadLibraryAlteredSearchPathFlag@@YAKXZ ; GetLoadLibraryAlteredSearchPathFlag(void)
.text:00000001800437B4  mov  rdi, [rsp+58h+hDll]
.text:00000001800437BC  mov  r8d, eax      ; dwFlags // the flag comes from the above call to GetLoadLibraryAlteredSearchPathFlag
.text:00000001800437BF  mov  r9, rdi      ; phMod
.text:00000001800437C2  mov  rdx, rbx       ; pwszFileName
.text:00000001800437C5                 call    ?LoadLibraryWithLogging@@YAJW4LoadOrFreeWhy@@@PEBGKPEAPEAUHINSTANCE__@@@Z ; LoadLibraryWithLogging(LoadOrFreeWhy,ushort const *,ulong,HINSTANCE __ **) // call to kernelbase!LoadLibraryExW
.text:00000001800437CA  mov  rcx, [rdi]
```
combbase!GetLoadLibraryAlteredSearchPathFlag gets the search path flag for DLL loading from a global variable of combase.dll.

.text:0000000180043944 ; unsigned int __fastcall GetLoadLibraryAlteredSearchPathFlag()
.text:0000000180043944
.text:0000000180043944
.text:0000000180043944          xchg    ax, ax
.text:0000000180043946          nop     dword ptr [rax+00000000h]
.text:000000018004394D          sub     rsp, 38h
.text:0000000180043951          mov     eax, cs:g_LoadLibraryAlteredSearchPathFlag  // the flag is kept in a global variable of combase.dll
.text:0000000180043957          cmp     eax, 0FFFFFFFFh
.text:000000018004395A          jz      short loc_180043965
.text:000000018004395C loc_18004395C:                          ; CODE XREF: GetLoadLibraryAlteredSearchPathFlag(void)+80j
.text:000000018004395C          add     rsp, 38h
.text:0000000180043960          jmp     __guard_ss_common_verify_stub
• combase!g_LoadLibraryAlteredSearchPathFlag sits in the data section of combase.dll, so it’s open for write. By default, this variable is set to 0, and changes it to 8 (LOAD_WITH_ALTERED_SEARCH_PATH) will let Silverlight plugin (npctrl.dll) to be loaded successfully.
Silverlight Browser Plugin Loaded

ModLoad: 00000000`75360000 00000000`754fa000  c:\Program Files\Microsoft Silverlight\5.1.50901.0\npctrl.dll

ntdll!NtMapViewOfSection+0x14:
00000000`75360000 00000000`754fa000  c:\Program Files\Microsoft Silverlight\5.1.50901.0\npctrl.dll

# Child-SP RetAddr : Args to Child : Call Site
00 00000000`764f3688 00000000`d48049ff : 00000000`00000000 00000000`00000000 00000000`00000038 00000000`0000007c : ntdll!NtMapViewOfSection+0x14
01 00000000`764f3690 00000000`d480476e : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : ntdll!LdrpMinimalMapModule+0x0d3
02 00000000`764f3720 00000000`d4802f33 : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 01d20413`4a069200 : ntdll!LdrpMapDllWithSectionHandle+0x0e1
03 00000000`764f3790 00000000`d4800189 : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : ntdll!LdrpMapDllNtFileName+0x2eb
04 00000000`764f3860 00000000`d48003b6 : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : ntdll!LdrpLoadDll+0xc0d
05 00000000`764f39e0 00000000`d4807466 : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : ntdll!LdrpLoadDllInternal+0x132
06 00000000`764f3a40 00000000`d48072f7 : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : ntdll!LdrpProcessWork+0x08e
07 00000000`764f3ac0 00000000`d48064dc : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : ntdll!LdrpLoadDll+0x10b
08 00000000`764f3c60 00000000`d18e2a94 : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : KERNELBASE!LoadLibraryExW+0x184
09 00000000`764f3dd6 00000000`c76b6630 : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : EShims\INS_EdgeCI\LoadLibrary::APiHook_LoadLibraryExW+0x64
0a 00000000`764f3dd0 00000000`c76b6634 : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : combase\LoadLibraryWithLogging+0x035
0b 00000000`764f3e10 00000000`d1d237ca : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : combase\IClassCache::CdllPathEntry::LoadDll+0x0e5e
RWX Memory Created By Silverlight Plugin
Locate RWX Memory
The Fix to CFG Bypass By Loading Silverlight

• Arbitrary Code Guard (ACG) prevents this CFG/DEP bypass in a generic way, however this mitigation doesn’t specifically resolve the issue of bypassing ActiveX restriction in Edge.
  • npctrl.dll can’t be loaded when ACG is on, because its PE file’s import table (.idata section) is somehow marked as RX at the compilation time.
  • Even when npctrl.dll is loaded, it’s unable to create RWX memory.

• In Windows 10 build 14986 (WIP slow ring), we are still able to make the exploit work via thread opt out (THREAD_DYNAMIC_CODE_ALLOW = 1).
Suggestions for Preventing Data-only Attack

• Never leave critical data unprotected!
  • If possible, apply write protection on the critical data page. Remove the write protection only when it needs to be updated, and be sure to lock the data page immediately after the updating finishes.
  • If for some reason the scheme above is not feasible (such as granularity, performance etc), the critical data can probably be stored in a dynamically allocated memory (ASLR enabled), and its address needs to be encrypted and stored separately.
  • For the encryption scheme mentioned above, the secret key of encryption has to be in a protected area, such as in kernel space, to prevent the access from user-mode. Moreover, the strength of encoding or encryption algorithm needs to enhanced to prevent brute-force attack.

• Always try verifying the integrity of critical data before using it. With such extra logic being introduced, many data-only attacks can be detected in their early stage.
Conclusion

• With the emergence of fine-grained CFI solution, the approach of calling function out-of-context will gradually lose its effectiveness.

• Today, application programs and operating systems have a lot of unprotected data, which can be leveraged to conduct powerful attack without the need of altering the program’s execution flow.

• Even if people have already been aware of the danger of data-only attacks, it’s still very difficult to prevent.
  • In some cases, it’s almost impossible to distinguish an attack from a legitimate access. Therefore, data-only attacks can’t always be resolved from the program’s logical perspective.
  • Due to performance consideration, OS/Application can’t move all its critical data into kernel space. In most cases, such user-space data will be protected by either “read-only” memory attribute (such as PE module’s import/export table section, .mrdata section of ntdll.dll) or simple encoding (RtlEncodePointer).
  • The battle of contending for the protected memory will continue.

• Data-only attack may have some variations, and it can be combined with some other exploitation techniques, such as race condition. We have discovered a couple of such bugs, and we would like to share the details after they are fixed by the vendor.
Q & A

• This concludes the part I of our data-only attack talk, in part II we’ll be presenting more examples of data-only attack in detail. Stay tuned!

• You are welcomed to send questions to
  • Bing Sun @ bing.sun@intel.com
  • Chong Xu @ chong.c.xu@intel.com
  • Stanley Zhu @ stanley.zhu@intel.com

• Thank MSRC for getting the issues fixed in a timely manner.

• Special thanks to Haifei Li and the McAfee IPS Vulnerability Research team.
References

• From read-write anywhere to controllable calls
• Mitigation bounty — 4 techniques to bypass mitigations
• Bypassing Control Flow Guard in Windows 10
• Use Chakra engine again to bypass CFG
• Chakra JIT CFG Bypass
• Bypass Control Flow Guard Comprehensively
• JIT Spraying Never Dies - Bypass CFG By Leveraging WARP Shader JIT Spraying
• Look Mom, I don’t use Shellcode
• Write Once, Pwn Anywhere