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

Bing Sun, Chong Xu

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.

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 Fundamental 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
  - Bypass CFG by calling ntdll!LdrResolveDelayLoadsFromDll
- Data-only Attack
  - Leak stack address from LdrpWork structure
  - Bypass CFG/DEP by abusing LdrpWork mechanism
- Suggestions for Preventing Data-only Attack
- Conclusion
- Acknowledgement

The Fundamental 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 Fundamental of CFG (Continued)

```
0:034> u RPCRT4!Ndr64pCommonStringUnmarshall+0x137
RPCRT4!Ndr64pCommonStringUnmarshall+0x137:
00007fff`67c34167 ff154b330e00    call    qword ptr [RPCRT4!_guard_dispatch_icall_fptr (00007fff`67d174b8)]
00007fff`67c3416d 488bd0          mov     rdx,rax
00007fff`67c34170 4885c0          test    rax,rax
00007fff`67c34173 0f845e3c0700    je      RPCRT4!Ndr64pCommonStringUnmarshall+0x73da7 (00007fff`67ca7dd7)
00007fff`67c34179 488d0c07        lea     rcx,[rdi+rax]
00007fff`67c3417d c7014c4d454d    mov     dword ptr [rcx],4D454D4Ch
00007fff`67c34183 897904          mov     dword ptr [rcx+4],edi
00007fff`67c34186 488b8338010000  mov     rax,qword ptr [rbx+138h]
0:034> dq 7fff67d174b8 L1
00007fff`67d174b8  00007fff`67ea5850
0:034> u 7fff67ea5850 Lf
ntdll!LdrpDispatchUserCallTarget:
00007fff`67ea5850 4c8b1db1ea0c00  mov     r11,qword ptr [ntdll!LdrSystemDllInitBlock+0x68 (00007fff`67f74308)]
00007fff`67ea5857 4c8bd0          mov     r10,rax
00007fff`67ea585a 49c1ea09        shr     r10,9
00007fff`67ea585e 4f8b1cd3        mov     r11,qword ptr [r11+r10*8]
00007fff`67ea5862 4c8bd0          mov     r10,rax
00007fff`67ea5865 49c1ea03        shr     r10,3
00007fff`67ea5869 a80f            test    al,0Fh
00007fff`67ea586b 7509            jne     ntdll!LdrpDispatchUserCallTarget+0x26 (00007fff`67ea5876)
00007fff`67ea586d 4d0fa3d3        bt      r11,r10
00007fff`67ea5871 7310            jae     ntdll!LdrpDispatchUserCallTarget+0x33 (00007fff`67ea5883)
00007fff`67ea5873 48ffe0          jmp     rax
00007fff`67ea5876 4983ca01        or      r10,1
00007fff`67ea587a 4d0fa3d3        bt      r11,r10
00007fff`67ea587e 7303            jae     ntdll!LdrpDispatchUserCallTarget+0x33 (00007fff`67ea5883)
00007fff`67ea5880 48ffe0          jmp     rax
```

16-byte aligned

Non 16-byte aligned, set bit 0 of offset

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

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)

Call Function Out-of-context

- 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.

The Delay Load Import Mechanism

- The Delay Load Import mechanism enables fast loading of a module by allowing its imported DLL to be loaded only on the first call to a function in that DLL.
- When the loading of an imported DLL is delayed, the imported functions in that DLL are temporarily pointing to some stub functions. Later when such an imported function is called, its stub function will use a helper function (__delayLoadHelper2) to load the imported DLL, and resolve the imported function to its real entry point at run time. Afterwards, the call to the imported function will happen naturally, without the intervention of stub and helper function.
- __delayLoadHelper2 calls the native API ntdll!ResolveDelayLoadedAPI to do the actual job. In order for ntdll!ResolveDelayLoadedAPI to know how to perform the delay load import, a critical data structure (DELAY_IMPORT_DESCRIPTOR) needs to be passed to it in the 2nd argument.

Example of Delay Load Import

```
WININET!_tailMerge_bcrypt_dll
WININET!_CreateFileWithRandomFilename+0x81
WININET!CreateUniqueCacheFile+0x289
WININET!CCacheClientFileManager::CreateUniqueFile+0xc8
WININET!CCacheClientContainer::CreateUniqueFile+0x65
```

Disassembly

Offset: @$scopeip

```
6a0fccb2 b824c21d6a      mov     eax,offset WININET!_imp__ntohl (6a1dc224)
6a0fccb7 e96cfeffff      jmp     WININET!_tailMerge_WS2_32_dll (6a0fcb28)
WININET!_imp_load__CertVerifyCertificateChainPolicy:
6a0fccbc b818c01d6a      mov     eax,offset WININET!_imp__CertVerifyCertificateChainPolicy (6a1dc018)
6a0fccc1 e996fdffff      jmp     WININET!_tailMerge_CRYPT32_dll (6a0fca5c)
WININET!_imp_load__BCryptCreateHash:
6a0fccc6 b82cc41d6a      mov     eax,offset WININET!_imp__BCryptCreateHash (6a1dc42c)
6a0fcccb e900000000      jmp     WININET!_tailMerge_bcrypt_dll (6a0fccd0)
WININET!_tailMerge_bcrypt_dll:
6a0fccd0 51              push    ecx
6a0fccd1 52              push    edx
6a0fccd2 50              push    eax
6a0fccd3 68c0361c6a      push    offset WININET!_DELAY_IMPORT_DESCRIPTOR_bcrypt_dll (6a1c36c0)
6a0fccd8 e8cd36ffff      call    WININET!__delayLoadHelper2 (6a0f03aa)
6a0fccdd 5a              pop     edx
6a0fccde 59              pop     ecx
6a0fccdf ffe0            jmp     eax
```

Delay load Helper function

Command

```
0:008> u poi(WININET!_imp__BCryptGenRandom) L2
WININET!_imp_load__BCryptGenRandom:
6a0fcdd8 b824c41d6a      mov     eax,offset WININET!_imp__BCryptGenRandom (6a1dc424)
6a0fcddd e9eefeffff      jmp     WININET!_tailMerge_bcrypt_dll (6a0fccd0)
```

Stub function

Example of Delay Load Import (Continued)

```
WININET!_tailMerge_bcrypt_dll+0xf
WININET!_CreateFileWithRandomFilename+0x81
WININET!CreateUniqueCacheFile+0x289
WININET!CCacheClientFileManager::CreateUniqueFile+0xc8
WININET!CCacheClientContainer::CreateUniqueFile+0x65
```

Disassembly

Offset: @$scopeip

```
WININET!_tailMerge_bcrypt_dll:
6a0fccd0 51              push    ecx
6a0fccd1 52              push    edx
6a0fccd2 50              push    eax
6a0fccd3 68c0361c6a      push    offset WININET!_DELAY_IMPORT_DESCRIPTOR_bcrypt_dll (6a1c36c0)
6a0fccd8 e8cd36ffff      call    WININET!__delayLoadHelper2 (6a0f03aa)
6a0fccdd 5a              pop     edx
6a0fccde 59              pop     ecx
6a0fccdf ffe0            jmp     eax {bcrypt!BCryptGenRandom (745b5160)}
WININET!_imp_load__BCryptHashData:
6a0fcce1 b834c41d6a      mov     eax,offset WININET!_imp__BCryptHashData (6a1dc434)
6a0fcce6 e9e5ffffff      jmp     WININET!_tailMerge_bcrypt_dll (6a0fccd0)
WININET!_imp_load__BCryptFinishHash:
6a0fcceb b830c41d6a      mov     eax,offset WININET!_imp__BCryptFinishHash (6a1dc430)
6a0fccf0 e9dbffffff      jmp     WININET!_tailMerge_bcrypt_dll (6a0fccd0)
WININET!_imp_load__BCryptDestroyHash:
6a0fccf5 b820c41d6a      mov     eax,offset WININET!_imp__BCryptDestroyHash (6a1dc420)
```

Jump to the real entry point

Command

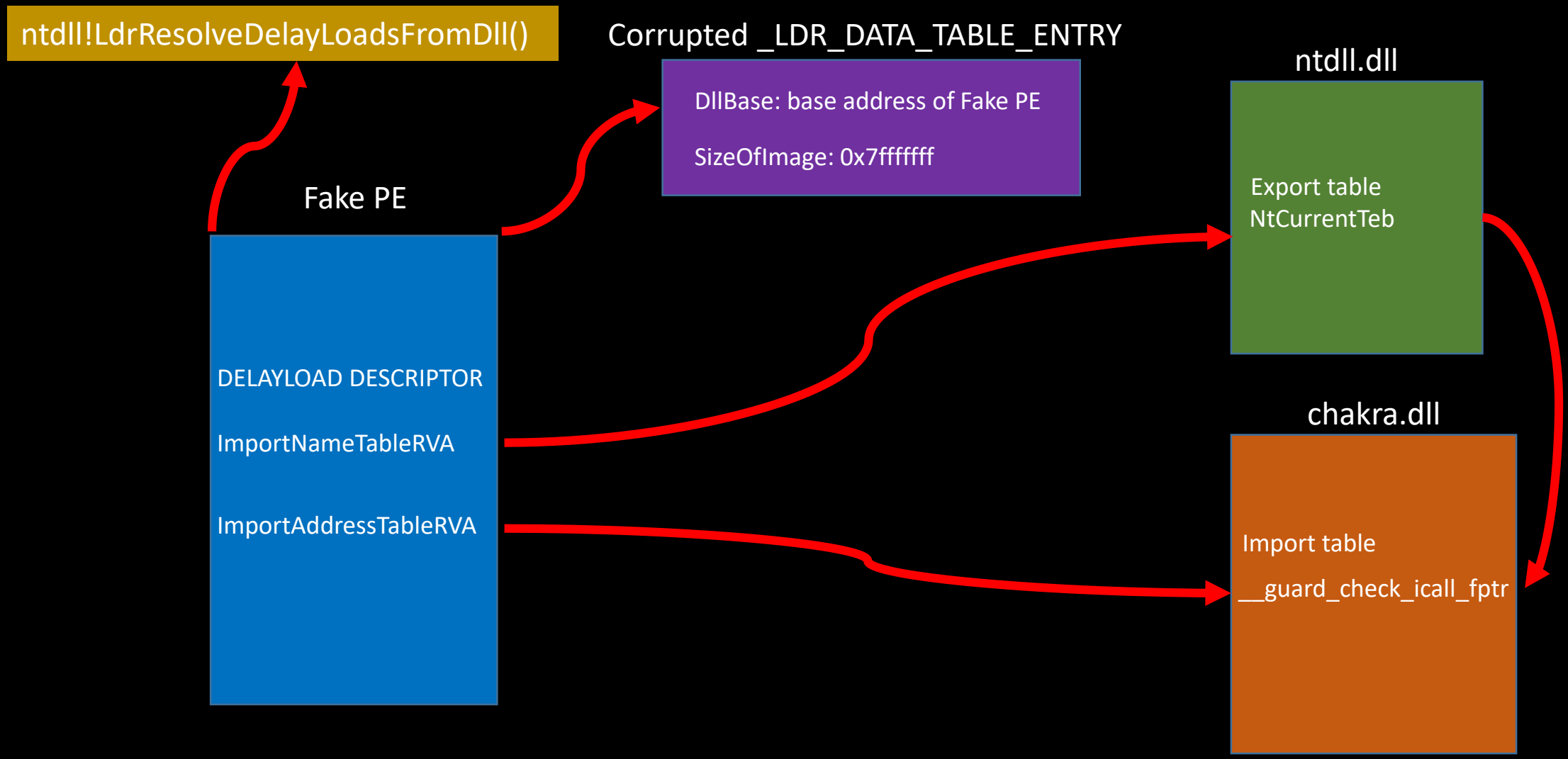
```
0:008> u poi(WININET!_imp__BCryptGenRandom) L2
bcrypt!BCryptGenRandom:
745b5160 8bff            mov     edi,edi
745b5162 55              push    ebp
```

Real entry point

Bypass CFG By Calling ntdll!LdrResolveDelayLoadsFromDll

- ntdll!LdrResolveDelayLoadedAPI and ntdll!LdrResolveDelayLoadsFromDll can be leveraged to overwrite any read-only function pointer, and both are valid for CFG (in WIP build 14986 and before). In terms of making out-of-context function call, the latter is much easier as it has fewer arguments to control.
- A fake PE carrying a malformed DELAY_IMPORT_DESCRIPTOR structure is created in memory and fed to ntdll!LdrResolveDelayLoadsFromDll. The ImportAddressTableRVA field of delay import descriptor points to the function pointer to be overwritten (could be outside the fake PE).
- In order for the operating system's module loader to believe the fake PE is a loaded module, some existing entry in LDR_DATA_TABLE needs to be corrupted (and will be restored later). In addition, the SizeOfImage field of corrupted entry needs to be set big enough to pass the range check.

The Diagram of CFG Bypass Scheme



The Original Call Stack of array.push Method

```
0:008> r
eax=5d9cdec0 ebx=12000180 ecx=12000180 edx=090e2820 esi=00000002 edi=12010000
eip=5d9cdec0 esp=051fce68 ebp=051fcea0 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00200246
chakra!Js::JavascriptNativeIntArray::SetItem:
5d9cdec0 55              push    ebp
0:008> kv
 # ChildEBP RetAddr  Args to Child
00 051fce64 5d9ba007 12010000 12012001 00000000 chakra!Js::JavascriptNativeIntArray::SetItem (FPO: [3,13,4])
01 051fcea0 5d9e3708 0b280160 10000002 12000180 chakra!Js::JavascriptArray::EntryPush+0x237 (FPO: [2,7,4])
02 051fcef0 5d9e56e5 093485be 0b280160 09320000 chakra!Js::InterpreterStackFrame::OP_CallCommon<Js::OpLayoutDynamicProfile<Js::OpLayoutT_CallI<Js::LayoutSizeF
03 051fcf18 5d9eb01f 093485be 051fd148 09320000 chakra!Js::InterpreterStackFrame::OP_ProfiledReturnTypeCallI<Js::OpLayoutT_CallI<Js::LayoutSizePolicy<0> > >+0
04 051fcf38 5d9e77d6 46632e9d 00000000 051fcfd0 chakra!Js::InterpreterStackFrame::ProcessProfiled+0xa9f (FPO: [0,1,4])
05 051fcf70 5d9e678a 46632e41 09348578 0934858e chakra!Js::InterpreterStackFrame::Process+0xa6 (FPO: [Non-Fpo])
06 051fcfac 5d9eb57d 0934858e 09320000 051fcfd0 chakra!Js::InterpreterStackFrame::OP_TryCatch+0x49 (FPO: [Non-Fpo])
07 051fcfc8 5d9ed611 09348000 093485c3 00000000 chakra!Js::InterpreterStackFrame::ProcessProfiled+0xffd (FPO: [0,1,4])
08 051fd238 5d9ed158 051fd268 00000000 00000000 chakra!Js::InterpreterStackFrame::InterpreterHelper+0x4a1 (FPO: [Non-Fpo])
09 051fd264 09360fe2 051fd278 051fd2b4 5da4fc91 chakra!Js::InterpreterStackFrame::InterpreterThunk+0x38 (FPO: [1,1,4])
WARNING: Frame IP not in any known module. Following frames may be wrong.
0a 051fd270 5da4fc91 09101950 00000000 00000000 0x9360fe2
0b 051fd2b4 5d9b1cfe 00000000 00000000 466332c1 chakra!Js::JavascriptFunction::CallFunction<1>+0x91 (FPO: [Non-Fpo])
0c 051fd32c 5d9b134b 04c19c10 00000000 00000000 chakra!Js::JavascriptFunction::CallRootFunctionInternal+0xee (FPO: [Non-Fpo])
0d 051fd384 5da3262c 04c19c10 00000000 00000000 chakra!Js::JavascriptFunction::CallRootFunction+0x38 (FPO: [Non-Fpo])
0e 051fd3e0 5d9b2ac5 051fd410 00000000 00000000 chakra!ScriptSite::CallRootFunction+0x5a (FPO: [Non-Fpo])
0f 051fd424 5d983b4e 09101950 051fd45c 00000000 chakra!ScriptSite::Execute+0x105 (FPO: [Non-Fpo])
10 051fd49c 5d98475e 051fd6f8 051fd708 466334dd chakra!ScriptEngine::ExecutePendingScripts+0x162 (FPO: [Non-Fpo])
11 051fd530 5d984904 0538a024 00000000 02f7e2e8 chakra!ScriptEngine::ParseScriptTextCore+0x2cd (FPO: [Non-Fpo])
12 051fd57c 5e5b7f7f 04c187fc 0538a024 00001df6 chakra!ScriptEngine::ParseScriptText+0x44 (FPO: [Non-Fpo])
13 051fd5d0 5e5b7e5d 0538a024 00000000 00000000 edgehtml!CJScript9Holder::ParseScriptText+0xa1 (FPO: [Non-Fpo])
14 051fd644 5e5b83b6 00000000 00000000 0538a024 edgehtml!CScriptCollection::ParseScriptText+0x151 (FPO: [19,9,4])
15 051fd728 5e5b59f4 02f72500 ffffffff 00000000 edgehtml!CScriptData::CommitCode+0x38b (FPO: [0,33,4])
16 051fd790 5e63e21d 02f72500 ffffffff 0526c150 edgehtml!CScriptData::Execute+0x22e (FPO: [0,21,4])
17 051fd7b0 5e5a9dc8 0526c150 02f78540 02f78540 edgehtml!CHtmScriptParseCtx::Execute+0x9d (FPO: [Non-Fpo])
18 051fd7f8 5e4f3bd9 5e64a0c0 02f78540 00000000 edgehtml!CHtmParseBase::Execute+0x148 (FPO: [0,13,4])
19 051fd81c 5e4f3834 00000002 0526c150 5e556240 edgehtml!CHtmPost::Broadcast+0x4d (FPO: [1,3,4])
1a 051fd93c 5e5561cb 00721fce 02f88000 02f78540 edgehtml!CHtmPost::Exec+0x364 (FPO: [1,65,4])
1b 051fd95c 5e5560c5 00721fce 02f78540 02f88000 edgehtml!CHtmPost::Run+0x3d (FPO: [Non-Fpo])
1c 051fd97c 5e555fba 02f78540 02f78540 5e1345a8 edgehtml!PostManExecute+0x60 (FPO: [1,2,0])
1d 051fd990 5e5570e8 5e5570b0 051fd9d0 00000007 edgehtml!PostManResume+0x7b (FPO: [0,0,4])
1e 051fd9c0 5e5adefd 02f96380 02f78540 02f88000 edgehtml!CHtmPost::OnDwnChanCallback+0x38 (FPO: [Non-Fpo])
1f 051fd9d4 5e5abd4c 00000000 00008002 02f88000 edgehtml!CDwnChan::OnMethodCall+0x3d (FPO: [1,0,0])
20 051fda80 5e5ab5b3 44af4a41 00008002 000b02f6 edgehtml!GlobalWndOnMethodCall+0x32c (FPO: [0,37,4])
21 051fdac4 763e3657 000b02f6 00008002 00000000 edgehtml!GlobalWndProc+0x153 (FPO: [Non-Fpo])
22 051fdaf0 763c136a 5e5ab460 000b02f6 00008002 user32!Ordinal2535+0x1037
23 051fdbc8 763c0e58 000b02f6 00008002 00000000 user32!DispatchMessageW+0x6ca
24 051fdc30 763c0cb0 688d97c2 051ffdcc 5b6796e1 user32!DispatchMessageW+0x1b8
25 051fdc3c 5b6796e1 051fdccc 40506a7b 0306e150 user32!DispatchMessageW+0x10
26 051ffdcc 5b6940f8 40506917 051ffeac 5b693de0 EdgeContent!CBrowserTab::_TabWindowThreadProc+0xa31 (FPO: [Non-Fpo])
27 051ffea0 6583c7cc 033069e0 6583c7b0 6583c7b0 EdgeContent!LCIETab_ThreadProc+0x318 (FPO: [Non-Fpo])
28 051ffeb8 761596c4 03304900 761596a0 6dcc32f7 msiso!CIsoScope::RegisterThread+0xbc (FPO: [Non-Fpo])
29 051ffecc 76ffe4a7 03304900 6cdb2f26 00000000 KERNEL32!BaseThreadInitThunk+0x24 (FPO: [Non-Fpo])
2a 051fff14 76ffe47b ffffffff 77045193 00000000 ntdll!__RtlUserThreadStart+0x2b (FPO: [Non-Fpo])
2b 051fff24 00000000 6583c7b0 03304900 00000000 ntdll!_RtlUserThreadStart+0x1b (FPO: [Non-Fpo])
```

Call ntdll!LdrResolveDelayLoadsFromDll Out-of-context

```
chakra!Windows::Data::Text::IUnicodeCharactersStatics::`vcall'{72}'+0x65eaf
chakra!Js::InterpreterStackFrame::OP_CallCommon<Js::OpLayoutDynamicProfile<Js::OpLayoutT_CallI<Js::LayoutSizePolicy<0> > > >+0x198
chakra!Js::InterpreterStackFrame::OP_ProfiledReturnTypeCallI<Js::OpLayoutT_CallI<Js::LayoutSizePolicy<0> > >+0x35
chakra!Js::InterpreterStackFrame::ProcessProfiled+0xa9f
chakra!Js::InterpreterStackFrame::Process+0xa6
chakra!Js::InterpreterStackFrame::OP_TryCatch+0x49
chakra!Js::InterpreterStackFrame::ProcessProfiled+0xffd
chakra!Js::InterpreterStackFrame::InterpreterHelper+0x4a1
chakra!Js::InterpreterStackFrame::InterpreterThunk+0x38
0x8f50fe2
chakra!Js::JavascriptFunction::CallFunction<1>+0x91
chakra!Js::JavascriptFunction::CallRootFunctionInternal+0xee
chakra!Js::JavascriptFunction::CallRootFunction+0x38
chakra!ScriptSite::CallRootFunction+0x5a
chakra!ScriptSite::Execute+0x105
```

out-of-context call via
chakra!Js::JavascriptArray::EntryPush

Disassembly

Offset: @$scopeip

```
5dac6a51 6a00            push    0
5dac6a53 ff7514          push    dword ptr [ebp+14h]
5dac6a56 57              push    edi
5dac6a57 ff15dc54de5d    call    dword ptr [chakra!__guard_check_icall_fptr (5dde54dc)]
5dac6a5d 8bcb            mov     ecx,ebx
5dac6a5f ff54242c        call    dword ptr [esp+2Ch]  ss:0023:04efcdb8={ntdll!LdrResolveDelayLoadsFromDll (7706b470)}
5dac6a63 3bf4            cmp     esi,esp
5dac6a65 7407            je      chakra!Windows::Data::Text::IUnicodeCharactersStatics::`vcall'{72}'+0x65ebe (5dac6a6e)
5dac6a67 b904000000      mov     ecx,4
5dac6a6c cd29            int     29h
```

Command

```
0:009> r
eax=0ee0d68e ebx=12000180 ecx=12000180 edx=00004000 esi=04efcd98 edi=12010000
eip=5dac6a5f esp=04efcd8c ebp=04efcdc0 iopl=0         nv up ei pl zr na pe cy
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00200247
chakra!Windows::Data::Text::IUnicodeCharactersStatics::`vcall'{72}'+0x65eaf:
5dac6a5f ff54242c        call    dword ptr [esp+2Ch]  ss:0023:04efcdb8={ntdll!LdrResolveDelayLoadsFromDll (7706b470)}
0:009> db poi(esp) L50
12010000  4d 5a 00 00 00 00 00 00-00 00 00 00 00 00 00 00  MZ..............
12010010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
12010020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
12010030  00 00 00 00 00 00 00 00-00 00 00 00 40 00 00 00  ............@...
12010040  50 45 00 00 00 00 00 00-00 00 00 00 00 00 00 00  PE..............
0:009> da poi(esp+4)
12012001  "ntdll.dll"
0:009> ? poi(esp+8)
Evaluate expression: 0 = 00000000
```

1st argument: fake PE

2nd argument: DLL name

3rd argument: 0

ntdll!LdrResolveDelayLoadsFromDll

Native API ntdll!LdrResolveDelayLoadsFromDll

```
.text:6A2CB470 ; int __stdcall LdrResolveDelayLoadsFromDll(PVOID ImageBase, int, int)
...
.text:6A2CB475          cmp     [ebp+arg_8], 0
.text:6A2CB479          jz      short loc_6A2CB482          //  The 3rd argument must be 0
...
.text:6A2CB482          mov     edx, [ebp+arg_4]
.text:6A2CB485          mov     ecx, [ebp+ImageBase] ; ImageBase
.text:6A2CB488          call    _LdrpGetDelayloadDescriptor@8 ; LdrpGetDelayloadDescriptor(x,x)   //  Get delay load descriptor from the fake PE
.text:6A2CB48D          test    eax, eax                    // return delay load descriptor
.text:6A2CB48F          jnz     short loc_6A2CB498
...
.text:6A2CB498          mov     ecx, [ebp+ImageBase]        // fake PE
.text:6A2CB49B          mov     edx, eax                    // delay load descriptor
.text:6A2CB49D          call    _LdrpResolveDelayLoadDescriptor@8 ; LdrpResolveDelayLoadDescriptor(x,x)
...
```

ntdll!LdrpResolveDelayLoadDescriptor

ntdll!LdrpResolveDelayLoadDescriptor

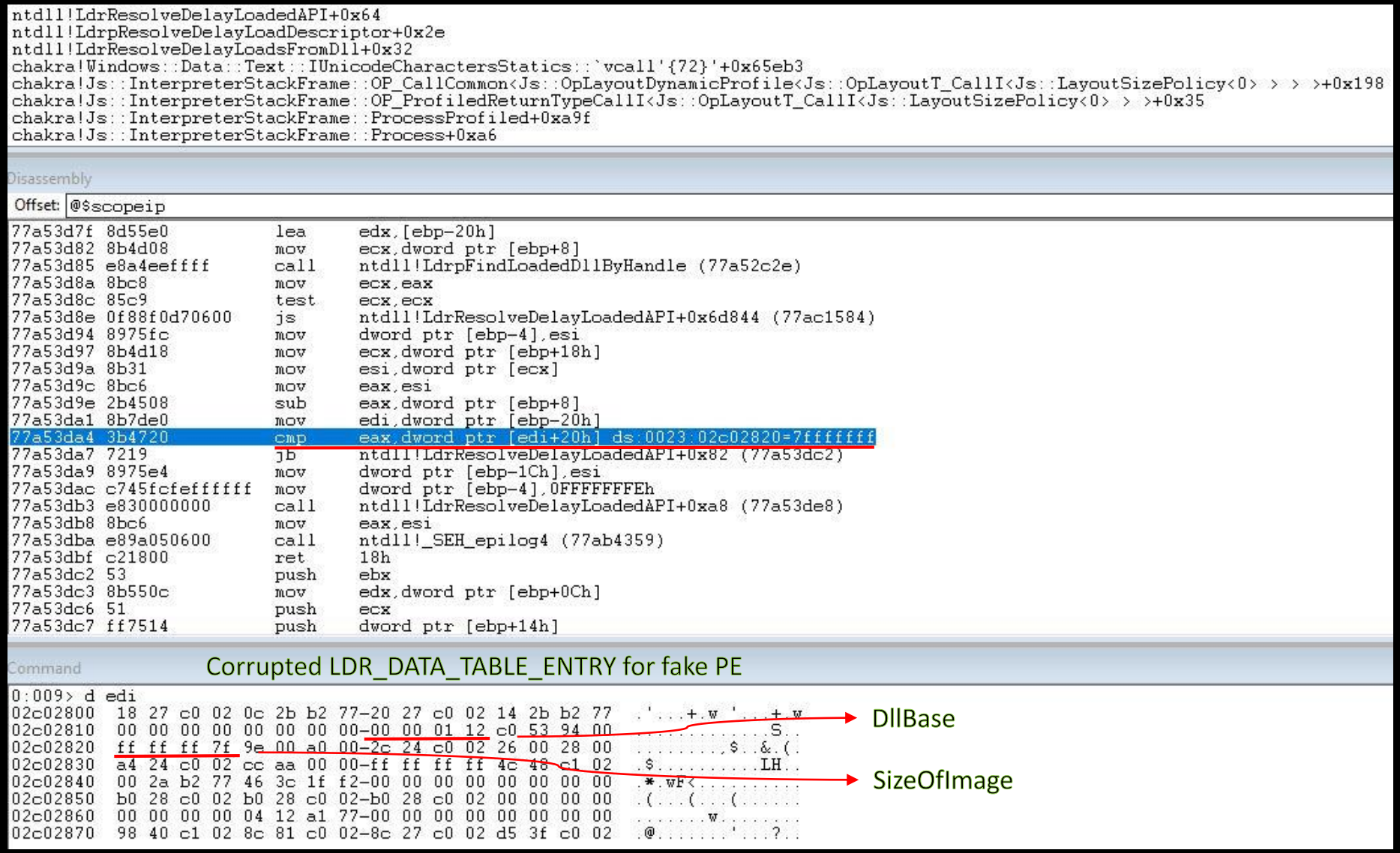
```
.text:6A243F9F ; __stdcall LdrpResolveDelayLoadDescriptor(x, x)
...
.text:6A243FA8          mov     ebx, [edx+0Ch]      // ImportAddressTableRVA
.text:6A243FAB          push    esi
.text:6A243FAC          add     ebx, ecx            // the address of function pointer to be overwritten
...
.text:6A243FB9          cmp     [ebx], esi
.text:6A243FBB          jz      short loc_6A243FE0          // the current function pointer must not be null
.text:6A243FBD          mov     eax, ebx
.text:6A243FBF          push    0
.text:6A243FC1          push    eax     // the address of function pointer to be overwritten
.text:6A243FC2          push    0
.text:6A243FC4          push    0
.text:6A243FC6          push    edx     // delay load descriptor
.text:6A243FC7          push    ecx     // fake PE
.text:6A243FC8          call    _LdrResolveDelayLoadedAPI@24 ; LdrResolveDelayLoadedAPI(x,x,x,x,x,x)
...
```

ntdll!LdrResolveDelayLoadedAPI

ntdll!LdrResolveDelayLoadedAPI

```
.text:6A243D40 ; __stdcall LdrResolveDelayLoadedAPI(x, x, x, x, x, x)
...
.text:6A243D7F          lea     edx, [ebp+Address]
.text:6A243D82          mov     ecx, [ebp+arg_0]                // fake PE
.text:6A243D85          call    _LdrpFindLoadedDllByHandle@12 ; LdrpFindLoadedDllByHandle(x,x,x)          // find LDR_DATA_TABLE_ENTRY of the fake PE
.text:6A243D8A          mov     ecx, eax
...
.text:6A243D97          mov     ecx, [ebp+arg_10]
.text:6A243D9A          mov     esi, [ecx]              // get the current function pointer
.text:6A243D9C          mov     eax, esi
.text:6A243D9E          sub     eax, [ebp+arg_0]    // delta = the current function pointer – fake PE
.text:6A243DA1          mov     edi, [ebp+Address]              // LDR_DATA_TABLE_ENTRY of the fake PE
.text:6A243DA4          cmp     eax, [edi+20h]      // delta must be less than SizeOfImage field of LDR_DATA_TABLE_ENTRY
.text:6A243DA7          jb      short loc_6A243DC2
...
.text:6A243DD8          call    _LdrpHandleProtectedDelayload@24 ; LdrpHandleProtectedDelayload(x,x,x,x,x,x)
```

Corrupt LDR Data Table to Pass the Range Check



Corrupted LDR_DATA_TABLE_ENTRY

```
0:009> lmvm microsoftedgecp
Browse full module list
start    end        module name
00940000 00956000   microsoftedgecp   (pdb symbols)          C:\ProgramData\dbg\sym\MicrosoftEdgeCP.pdb\62BD80CD01BC14B1ACDF30FE144B80B81\MicrosoftEdgeCP
    Loaded symbol image file: C:\Windows\SystemApps\Microsoft.MicrosoftEdge_8wekyb3d8bbwe\microsoftedgecp.exe
    Image path: C:\Windows\SystemApps\Microsoft.MicrosoftEdge_8wekyb3d8bbwe\microsoftedgecp.exe
    Image name: microsoftedgecp.exe
    Browse all global symbols  functions  data
    Timestamp:        ***** Invalid (F21F3C46)
    CheckSum:         00017F58
    ImageSize:        00016000
    File version:     11.0.14971.1000
    Product version:  11.0.14971.1000
    File flags:       0 (Mask 3F)
    File OS:          40004 NT Win32
    File type:        1.0 App
    File date:        00000000.00000000
    Translations:     0409.04b0
    CompanyName:      Microsoft Corporation
    ProductName:      Microsoft Edge
    InternalName:     MicrosoftEdgeCP
    OriginalFilename: MicrosoftEdgeCP.exe
    ProductVersion:   11.00.14971.1000
    FileVersion:      11.00.14971.1000 (rs_prerelease.161111-1700)
    FileDescription:  Microsoft Edge Content Process
    LegalCopyright:   © Microsoft Corporation. All rights reserved.
0:009> dt _LDR_DATA_TABLE_ENTRY 2c02800
ntdll!_LDR_DATA_TABLE_ENTRY
   +0x000 InLoadOrderLinks : _LIST_ENTRY [ 0x2c02718 - 0x77b22b0c ]
   +0x008 InMemoryOrderLinks : _LIST_ENTRY [ 0x2c02720 - 0x77b22b14 ]
   +0x010 InInitializationOrderLinks : _LIST_ENTRY [ 0x0 - 0x0 ]
   +0x018 DllBase          : 0x12010000 Void
   +0x01c EntryPoint       : 0x009453c0 Void
   +0x020 SizeOfImage      : 0x7fffffff
   +0x024 FullDllName      : _UNICODE_STRING "C:\Windows\SystemApps\Microsoft.MicrosoftEdge_8wekyb3d8bbwe\microsoftedgecp.exe"
   +0x02c BaseDllName      : _UNICODE_STRING "microsoftedgecp.exe"
   +0x034 FlagGroup        : [4]  "???"
   +0x034 Flags            : 0xaacc
```

ntdll!LdrpHandleProtectedDelayload

ntdll!LdrpHandleProtectedDelayload

```
.text:6A241C60 ; __stdcall LdrpHandleProtectedDelayload(x, x, x, x, x, x)
...
.text:6A241CCD          push    edi                     // the address of function pointer to be overwritten
.text:6A241CCE          push    [ebp+arg_C]
.text:6A241CD1          lea     eax, [ebp+var_228]
.text:6A241CD7          push    eax
.text:6A241CD8          call    _LdrpGetDelayloadExportDll@20 ; LdrpGetDelayloadExportDll(x,x,x,x,x)        // get LDR_DATA_TABLE_ENTRY of ntdll.dll
.text:6A241CDD          mov     [ebp+StatusCode], eax
.text:6A241CE3          test    eax, eax
.text:6A241CE5          js      loc_6A241FA2
.text:6A241CEB          mov     ecx, [ebp+var_228]      // LDR_DATA_TABLE_ENTRY of ntdll.dll
.text:6A241CF1          mov     ecx, [ecx+18h]  ; PcValue       // DllBase of ntdll.dll
.text:6A241CF4          call    _RtlGuardCheckImageBase@4 ; RtlGuardCheckImageBase(x)
.text:6A241CF9          mov     eax, [esi+18h]
...
```

Malformed IMAGE_DELAYLOAD_DESCRIPTOR

```
0:009> dt _IMAGE_DELAYLOAD_DESCRIPTOR 12010140
twinapi!_IMAGE_DELAYLOAD_DESCRIPTOR
   +0x000 Attributes       : _IMAGE_DELAYLOAD_DESCRIPTOR::<unnamed-type-Attributes>
   +0x004 DllNameRVA       : 0x180
   +0x008 ModuleHandleRVA  : 0x160
   +0x00c ImportAddressTableRVA : 0x4ce654dc
   +0x010 ImportNameTableRVA : 0x168
   +0x014 BoundImportAddressTableRVA : 0
   +0x018 UnloadInformationTableRVA : 0
   +0x01c TimeDateStamp    : 0
0:009> da 12010000 + 180
12010180  "ntdll.dll"
0:009> db 12010000 + 160 L4
12010160  00 00 00 00                                      ....
0:009> ln 12010000 + 4ce654dc
Browse module
Set bu breakpoint

(5ee754dc)   chakra!__guard_check_icall_fptr   |  (5ee754e0)   chakra!_IMPORT_DESCRIPTOR_msvcrt
Exact matches:
    chakra!__guard_check_icall_fptr = <no type information>
0:009> db 12010000 + 168 L4
12010168  39 01 00 80                                      9...
```

Exported entry 0x139 NtCurrentTeb in ntdll.dll

ntdll!LdrpHandleProtectedDelayload (Continued)

ntdll!LdrpHandleProtectedDelayload

```
...
.text:6A241DE3          lea     ecx, [ebp+var_230]
.text:6A241DE9          push    ecx
.text:6A241DEA          push    0
.text:6A241DEC          push    eax             // 313, the ordinal of the function to be resolved
.text:6A241DED          mov     ecx, [ebp+var_228]      // LDR_DATA_TABLE_ENTRY of ntdll.dll
.text:6A241DF3          call    _LdrpResolveProcedureAddress@20 ; LdrpResolveProcedureAddress(x,x,x,x,x)      // resolve function by its ordinal
...
.text:6A241E7B          push    eax
.text:6A241E7C          push    edi
.text:6A241E7D          push    [ebp+Address]     // the address of resolved function pointer
.text:6A241E83          mov     edx, [ebp+var_23C]      // the address of function pointer to be overwritten
.text:6A241E89          mov     ecx, esi        // LDR_DATA_TABLE_ENTRY of the fake PE
.text:6A241E8B          call    _LdrpWriteBackProtectedDelayLoad@20 ; LdrpWriteBackProtectedDelayLoad(x,x,x,x,x)          // overwrite the function pointer
...
```

ntdll!LdrpWriteBackProtectedDelayLoad

ntdll! LdrpWriteBackProtectedDelayLoad

```
.text:6A2742D7 ; __stdcall LdrpWriteBackProtectedDelayLoad(x, x, x, x, x)
...
.text:6A274310          push    4           ; NewProtect        // remove write protection on the function pointer
.text:6A274312          lea     eax, [ebp+ProtectSize]
.text:6A274315          push    eax         ; ProtectSize
...
.text:6A27431C          call    _ZwProtectVirtualMemory@20 ; ZwProtectVirtualMemory(x,x,x,x,x)
...
.text:6A274338          mov     [edi+eax], ecx                  // overwrite the function pointer
...
.text:6A274343          push    2           ; NewProtect        // restore write protection on the function pointer
.text:6A274345          lea     eax, [ebp+ProtectSize]
.text:6A274348          push    eax         ; ProtectSize
...
.text:6A27434F          call    _ZwProtectVirtualMemory@20 ; ZwProtectVirtualMemory(x,x,x,x,x)
...
```

Corrupted chakra!__guard_check_icall_fptr

```
ntdll!LdrpWriteBackProtectedDelayLoad+0x7d
ntdll!LdrpHandleProtectedDelayload+0x230
ntdll!LdrResolveDelayLoadedAPI+0x9d
ntdll!LdrpResolveDelayLoadDescriptor+0x2e
ntdll!LdrResolveDelayLoadsFromDll+0x32
```

Disassembly

Offset: @$scopeip

```
77a8434d 6aff            push    0FFFFFFFFh
77a8434f e86ccb0100      call    ntdll!NtProtectVirtualMemory (77aa0ec0)
77a84354 ff75f0          push    dword ptr [ebp-10h]  ss:0023:04efcb50=02c0284c
77a84357 e8a478fcff      call    ntdll!RtlReleaseSRWLockExclusive (77a4bc00)
77a8435c 5f              pop     edi
```

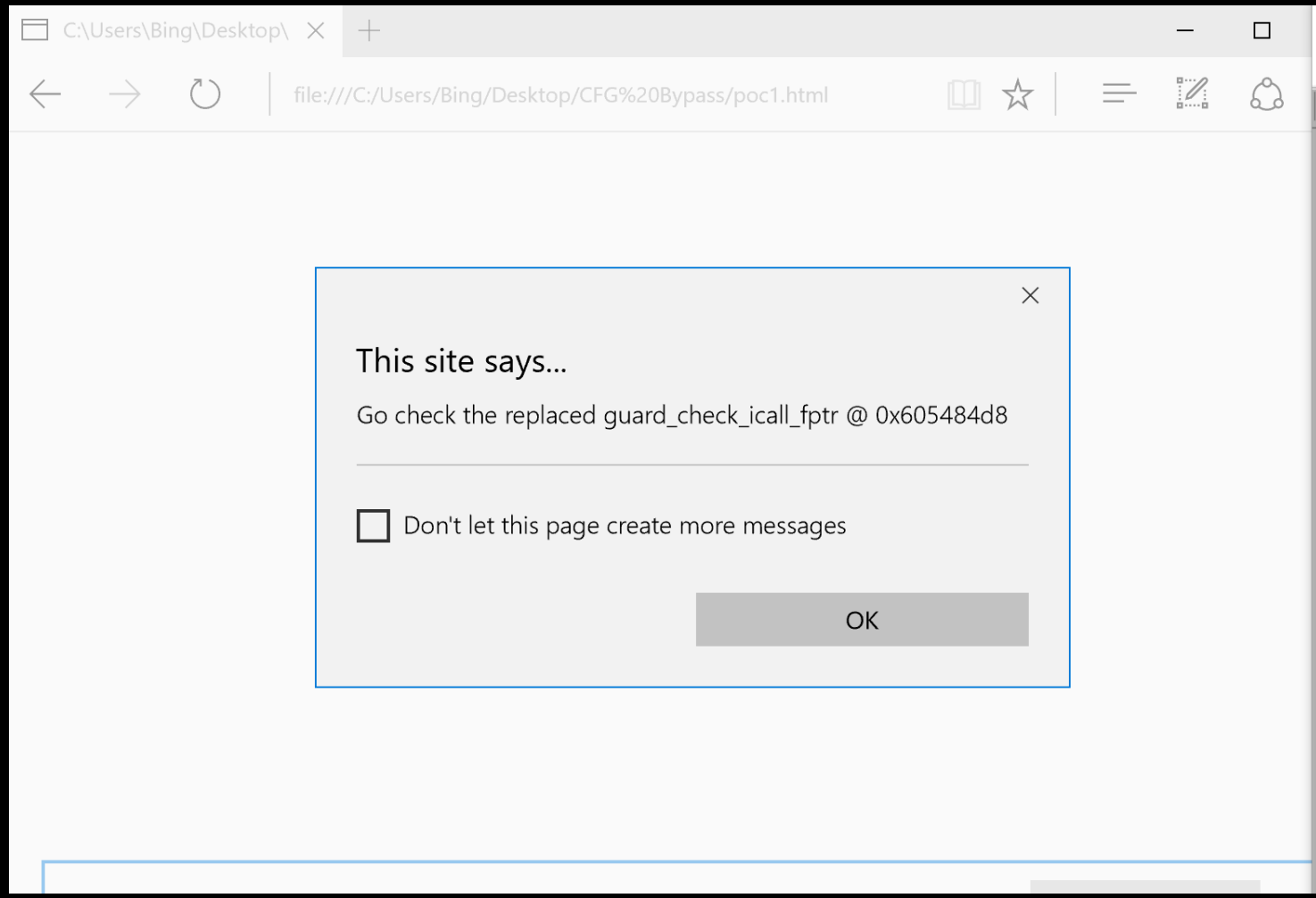
Command

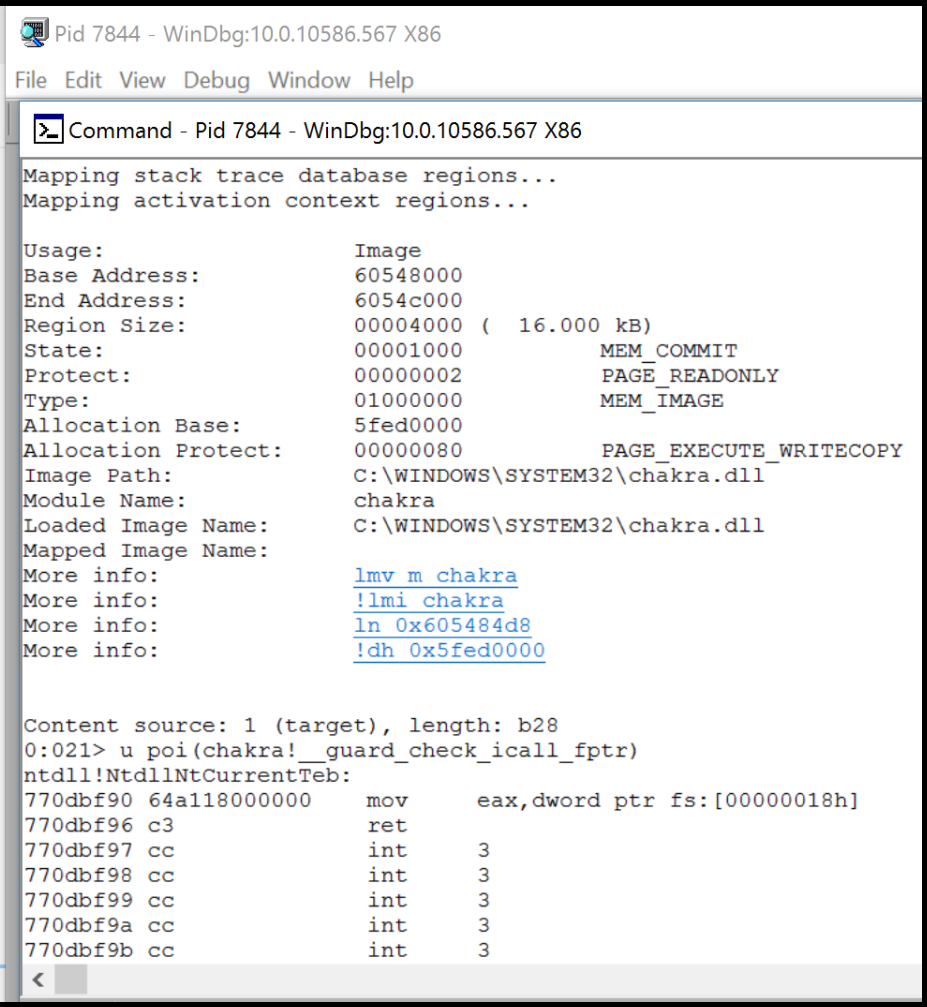
```
0:009> u poi(chakra!__guard_check_icall_fptr) L2
ntdll!NtdllNtCurrentTeb:
77add080 64a118000000    mov     eax,dword ptr fs:[00000018h]
77add086 c3              ret
0:009> !address chakra!__guard_check_icall_fptr

Usage:                  Image
Base Address:           5ee75000
End Address:            5eeca000
Region Size:            00055000 ( 340.000 kB)
State:                  00001000          MEM_COMMIT
Protect:                00000002          PAGE_READONLY
Type:                   01000000          MEM_IMAGE
Allocation Base:        5e830000
Allocation Protect:     00000080          PAGE_EXECUTE_WRITECOPY
Image Path:             C:\WINDOWS\SYSTEM32\chakra.dll
```

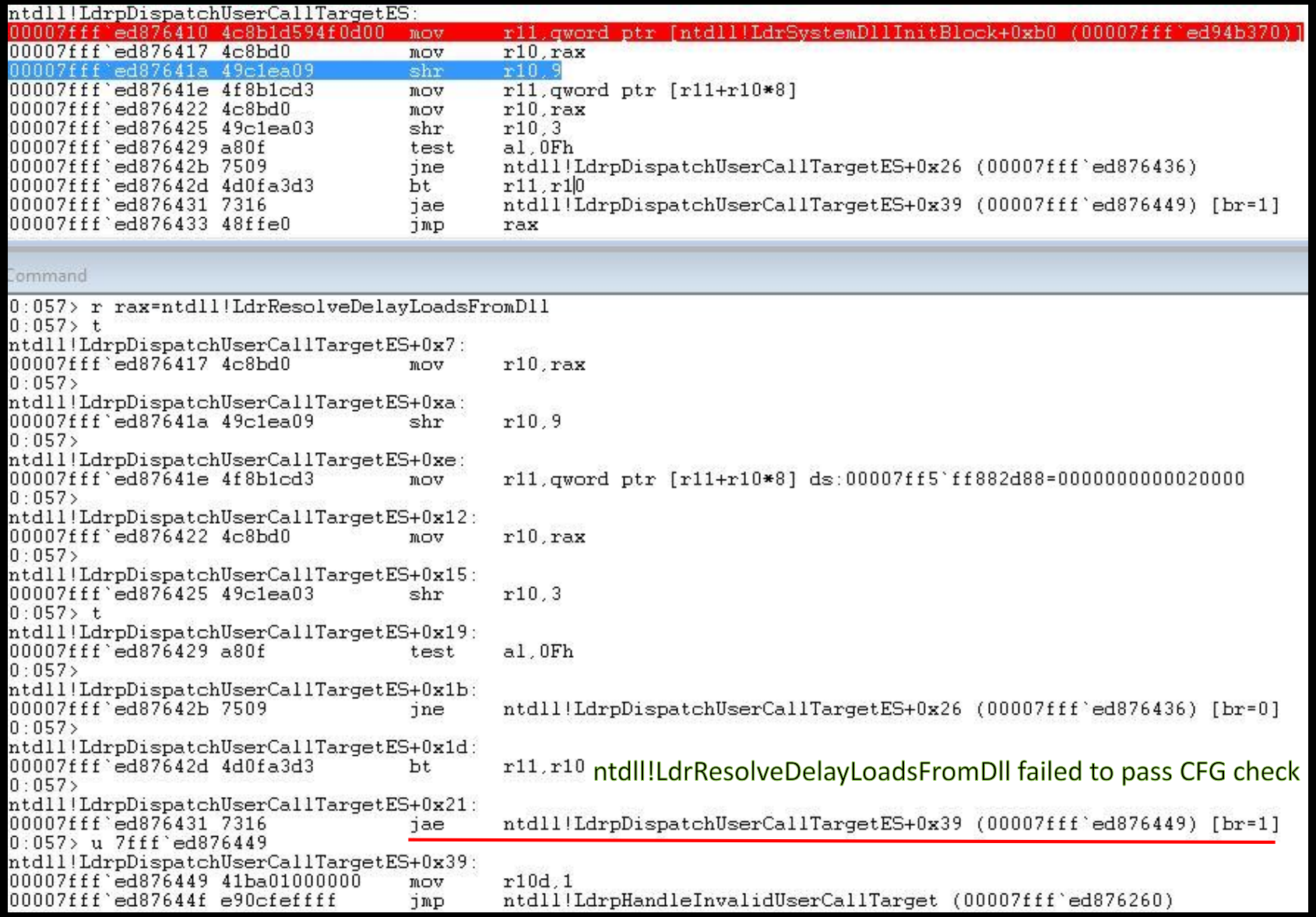
Exported entry 313 NtCurrentTeb in ntdll.dll

Bypass CFG By Calling ntdll!LdrResolveDelayLoadsFromDll





CFG Improvement in WIP 15048



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"

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 viare turn 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 fundamental of data-only attack
  - Change the program's default behavior by manipulating the data it depends on without divert 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 locate 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.

LdrpWork Mechanism

- The LdrpWork mechanism is responsible for performing various loader works related to the loading of a module, and these works can be processed either synchronously or asynchronously. The asynchronous mode is specifically used to support “parallel loading” of dependent modules.
- In memory, a LdrpWork structure contains a fixed size header and a variable length part (module name). In the asynchronous mode, all LdrpWork instances are organized using a FIFO doubly-linked list.
- LdrpWork mechanism consists of four major functions:
  - ntdll!LdrpAllocatePlaceHolder: allocate a new work from the loader’s private heap
  - ntdll!LdrpQueueWork: insert a new work at the tail of list
  - ntdll!LdrpDrainWorkQueue: remove the head node and send it to process routine
  - ntdll!LdrpProcessWork: process one single work, such as module mapping or snapping. This function can be invoked in both synchronous and asynchronous mode
- “Snapping” is one of the important loader works, and it’s fulfilled by:
  - ntdll!LdrpSnapModule: load dependent modules and resolve imported functions
  - ntdll!LdrpDoPostSnapWork: restore the memory attribute of .idata section

ntdll!LdrpAllocatePlaceHolder

ntdll!LdrpAllocatePlaceHolder

```
.text:00000001800158EC LdrpAllocatePlaceHolder proc near
...
.text:0000000180015911          mov     edx, cs:NtdllBaseTag
.text:0000000180015917          mov     r14d, r8d
.text:000000018001591A          add     edx, 40000h
.text:0000000180015920          mov     rsi, rcx    // pointer to UNICODE_STRING
.text:0000000180015923          xor     r12d, r12d
.text:0000000180015926          or      edx, 8      // Flags
.text:0000000180015929          mov     [rdi], r12
.text:000000018001592C          mov     ebp, r9d
.text:000000018001592F          movzx   r8d, word ptr [rcx]   // the path length of the module being loaded
.text:0000000180015933          mov     rcx, cs:LdrpHeap      // Heap handle
.text:000000018001593A          add     r8, 0B2h              // add the size of LdrpWork header (0xB0) and a null terminator (2)
.text:0000000180015941          call    RtlAllocateHeap
...
.text:0000000180015A00 LdrpAllocatePlaceHolder endp
```

ntdll!LdrpQueueWork

ntdll!LdrpQueueWork

```
.text:0000000180077260 LdrpQueueWork   proc near
...
.text:0000000180077275          lea     rcx, LdrpWorkQueueLock
.text:000000018007727C          call    RtlEnterCriticalSection
.text:0000000180077281          mov     rcx, cs:qword_1801542D8     // the tail of LdrpWorkQueue
.text:0000000180077288          lea     rdx, LdrpWorkQueue          // the head of LdrpWorkQueue
.text:000000018007728F          lea     rax, [rbx+38h]              // LIST_ENTRY field of LdrpWork structure
.text:0000000180077293          cmp     [rcx], rdx                  // verify the integrity of the doubly-linked list
.text:0000000180077296          jnz     loc_1800C79A8
.text:000000018007729C          mov     [rax+8], rcx                // insert the new LdrpWork at the tail of LdrpWorkQueue
.text:00000001800772A0          mov     [rax], rdx
.text:00000001800772A3          mov     [rcx], rax
.text:00000001800772A6          lea     rcx, LdrpWorkQueueLock
.text:00000001800772AD          mov     cs:qword_1801542D8, rax
.text:00000001800772B4          call    RtlLeaveCriticalSection
...
.text:00000001800772E7 LdrpQueueWork   endp
```

ntdll!LdrpDrainWorkQueue

ntdll!LdrpDrainWorkQueue

```
.text:0000000180003B3A0 LdrpDrainWorkQueue proc near
...
.text:0000000180003B3BF          lea     r12, LdrpWorkQueue
...
.text:0000000180003B3F1          mov     rbx, cs:LdrpWorkQueue      // get the 1st node from the head of LdrpWorkQueue
.text:0000000180003B3F8          mov     rax, [rbx]                 // get the 2nd node
.text:0000000180003B3FB          cmp     [rbx+8], r12               // verify the integrity of the doubly-linked list
.text:0000000180003B3FF          jnz     loc_1800BAD6C
.text:0000000180003B405          cmp     [rax+8], rbx               // verify the integrity of the doubly-linked list
.text:0000000180003B409          jnz     loc_1800BAD6C
.text:0000000180003B40F          mov     cs:LdrpWorkQueue, rax      // remove the 1st node from LdrpWorkQueue
.text:0000000180003B416          mov     [rax+8], r12
.text:0000000180003B41A          cmp     r12, rbx
.text:0000000180003B41D          jnz     loc_18003B4F7              // make sure the removed node is a LdrpWork to be processed
.text:0000000180003B423          cmp     cs:LdrpWorkInProgress, edi
.text:0000000180003B429          jnz     short loc_18003B438
```

ntdll!LdrpDrainWorkQueue (Continued)

ntdll!LdrpDrainWorkQueue

```
.text:000000018003B42B              mov     cs:LdrpWorkInProgress, 1
.text:000000018003B435              mov     sil, 1
...
.text:000000018003B444              test    sil, sil
.text:000000018003B447              jz      loc_18003B4E0
.text:000000018003B44D              test    edi, edi
.text:000000018003B44F              jnz     short loc_18003B47F
...
.text:000000018003B4E0              cmp     r12, rbx
.text:000000018003B4E3              jnz     short loc_18003B50C  // make sure the removed node is a LdrpWork to be processed
...
.text:000000018003B50C              lea     rcx, [rbx-38h]           // LdrpWork structure to be processed
.text:000000018003B510              mov     dl, bpl
.text:000000018003B513              call    LdrpProcessWork      // Process the LdrpWork node that is just removed from the head of LdrpWorkQueue
.text:000000018003B518              jmp     loc_18003B3D5
.text:000000018003B518 LdrpDrainWorkQueue endp
```

ntdll!LdrpProcessWork

ntdll!LdrpProcessWork

```
.text:0000000180010328 LdrpProcessWork proc near
...
.text:000000018001033B          mov     rbx, rcx
.text:000000018001033E          mov     rax, [rcx+20h]
.text:0000000180010342          cmp     dword ptr [rax], 0
.text:0000000180010345          jl      loc_1800104AD
.text:000000018001034B          mov     rax, [rcx+30h]          // _LDR_DATA_TABLE_ENTRY of current module being loaded
.text:000000018001034F          mov     rcx, [rax+98h]          // _LDR_DDAG_NODE
.text:0000000180010356          cmp     dword ptr [rcx+38h], 0          // test State._LDR_DDAG_NODE to determine which work to do
.text:000000018001035A          mov     rcx, rbx
.text:000000018001035D          jz      short loc_180010373
.text:000000018001035F          call    LdrpSnapModule          // load dependent modules and resolved imported functions
...
.text:0000000180010373          test    dword ptr [rbx+18h], 100000h
.text:000000018001037A          jnz     loc_18001042F
.text:0000000180010380          test    dword ptr [rbx+18h], 200h
.text:0000000180010387          jnz     short loc_1800103B1
.text:0000000180010389          call    LdrpMapDllSearchPath
```

ntdll!LdrpProcessWork (Continued)

ntdll!LdrpProcessWork

```
.text:000000018001038E          mov     edi, eax
.text:0000000180010390          mov     [rsp+58h+arg_10], eax
.text:0000000180010394          mov     ecx, 80000000h
.text:0000000180010399          lea     eax, [rdi+rcx]
.text:000000018001039C          test    ecx, eax
.text:000000018001039E          jnz     loc_1800104AD
...
.text:00000001800103B1          call    LdrpMapDllFullPath
.text:00000001800103B6          jmp     short loc_18001038E
...
.text:000000018001042F          call    LdrpMapDllRetry
.text:0000000180010434          mov     edi, eax
.text:0000000180010436          mov     [rsp+58h+arg_10], eax
.text:000000018001043A          jmp     loc_180010394
...
.text:000000018001050D LdrpProcessWork endp
```

ntdll!LdrpSnapModule

ntdll!LdrpSnapModule

```
.text:0000000180031940 LdrpSnapModule  proc near
...
.text:00000001800319A0          mov     eax, [rdi+70h]          // the number of already loaded dependent modules
.text:00000001800319A3          cmp     eax, [rdi+58h]          // the total number of all dependent modules
.text:00000001800319A6          jnb     loc_1800320FC
...                                                             // load each dependent module and resolve imported functions
.text:0000000180031D07          inc     dword ptr [rdi+70h]     // increase the counter
.text:0000000180031D0A          jmp     loc_1800319A0
...
.text:00000001800320FC          test    esi, esi
.text:00000001800320FE          js      loc_1800322B7
.text:0000000180032104          mov     rcx, rdi
.text:0000000180032107          call    LdrpDoPostSnapWork              // restore .idata section's memory attribute
...
.text:00000001800322C9 LdrpSnapModule  endp
```

ntdll!LdrpDoPostSnapWork

ntdll!LdrpDoPostSnapWork

```
.text:000000018004C30C LdrpDoPostSnapWork proc near
...
.text:000000018004C316          mov     rbx, rcx
.text:000000018004C319          xor     edi, edi
.text:000000018004C31B          mov     ecx, edi
.text:000000018004C31D          lea     rdx, [rbx+60h]          // the start address of .idata section
.text:000000018004C321          cmp     [rdx], rdi
.text:000000018004C324          jz      short loc_18004C34A
.text:000000018004C326          mov     r9d, [rbx+80h]          // the original memory attribute of .idata section in PE file
.text:000000018004C32D          lea     rax, [rsp+38h+arg_0]
.text:000000018004C332          lea     r8, [rbx+68h]           // the size of .idata section
.text:000000018004C336          mov     [rsp+38h+var_18], rax
.text:000000018004C33B          or      rcx, 0FFFFFFFFFFFFFFFFh
.text:000000018004C33F          call    ZwProtectVirtualMemory          //restore the .idata section's original memory attribute
...
```

ntdll!LdrpDoPostSnapWork (Continued)

ntdll!LdrpDoPostSnapWork

```
...
.text:000000018004C34A          mov     rax, [rbx+90h]
.text:000000018004C351          test    rax, rax
.text:000000018004C354          jz      short loc_18004C366
.text:000000018004C356          mov     rax, [rax]
.text:000000018004C359          cmp     rax, [rbx+88h]          // make sure _guard_check_icall_fptr is correctly resolved
.text:000000018004C360          jnz     loc_1800BD33C
.text:000000018004C366          mov     rax, [rbx+0A8h]
.text:000000018004C36D          test    rax, rax
.text:000000018004C370          jz      short loc_18004C382
.text:000000018004C372          mov     rax, [rax]
.text:000000018004C375          cmp     rax, [rbx+0A0h]         // make sure _guard_ss_verify_failure_fptr is correctly resolved
.text:000000018004C37C          jnz     loc_1800BD343
...
text:000000018004C3A6 LdrpDoPostSnapWork endp
```

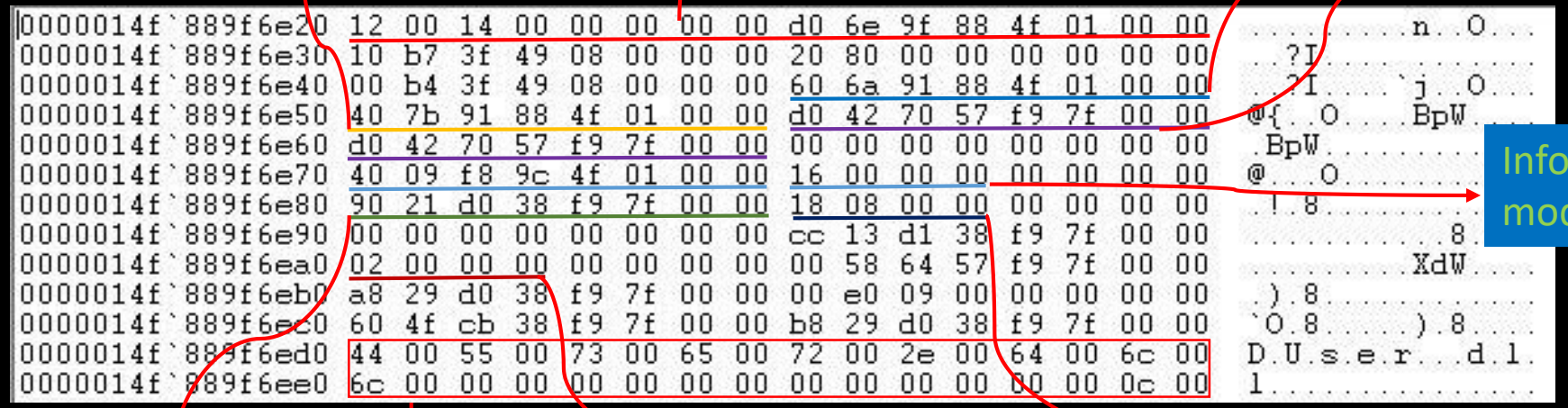
An Instance of LdrpWork in Memory

UNICODE_STRING: module name

_LDR_DATA_TABLE_ENTRY: parent module

_LDR_DATA_TABLE_ENTRY: module

LIST_ENTRY



Info about dependent modules

start address of .idata section

memory attribute of .idata section

size of .idata section

variable length part: module name

_LDR_DATA_TABLE_ENTRY

```
0:018> dt _LDR_DATA_TABLE_ENTRY 14f88917b40
Windows_UI!_LDR_DATA_TABLE_ENTRY
   +0x000 InLoadOrderLinks : _LIST_ENTRY [ 0x00007ff9`577043f0 - 0x0000014f`88916a60 ]
   +0x010 InMemoryOrderLinks : _LIST_ENTRY [ 0x00007ff9`57704400 - 0x0000014f`88916a70 ]
   +0x020 InInitializationOrderLinks : _LIST_ENTRY [ 0x00000000`00000000 - 0x00000000`00000000 ]
   +0x030 DllBase          : 0x00007ff9`38c90000 Void
   +0x038 EntryPoint       : 0x00007ff9`38cb3d10 Void
   +0x040 SizeOfImage      : 0x9e000
   +0x048 FullDllName      : _UNICODE_STRING "C:\Windows\SYSTEM32\DUser.dll"
   +0x058 BaseDllName      : _UNICODE_STRING "DUser.dll"
   +0x068 FlagGroup        : [4]  "???"
   +0x068 Flags            : 0xa2c4
   +0x068 PackagedBinary   : 0y0
   +0x068 MarkedForRemoval : 0y0
   +0x068 ImageDll         : 0y1
   +0x068 LoadNotificationsSent : 0y0
   +0x068 TelemetryEntryProcessed : 0y0
   +0x068 ProcessStaticImport : 0y0
   +0x068 InLegacyLists    : 0y1
   +0x068 InIndexes        : 0y1
   +0x068 ShimDll          : 0y0
   +0x068 InExceptionTable : 0y1
   +0x068 ReservedFlags1   : 0y00
   +0x068 LoadInProgress   : 0y0
   +0x068 LoadConfigProcessed : 0y1
   +0x068 EntryProcessed   : 0y0
   +0x068 ProtectDelayLoad : 0y1
   +0x068 ReservedFlags3   : 0y00
   +0x068 DontCallForThreads : 0y0
   +0x068 ProcessAttachCalled : 0y0
   +0x068 ProcessAttachFailed : 0y0
   +0x068 CorDeferredValidate : 0y0
   +0x068 CorImage         : 0y0
   +0x068 DontRelocate     : 0y0
   +0x068 CorILOnly        : 0y0
   +0x068 ReservedFlags5   : 0y000
   +0x068 Redirected       : 0y0
   +0x068 ReservedFlags6   : 0y00
   +0x068 CompatDatabaseProcessed : 0y0
   +0x06c ObsoleteLoadCount : 6
   +0x06e TlsIndex         : 0
   +0x070 HashLinks        : _LIST_ENTRY [ 0x00007ff9`57704070 - 0x0000014f`9ac263d0 ]
   +0x080 TimeDateStamp    : 0x1741d3cf
   +0x088 EntryPointActivationContext : (null)
   +0x090 Lock             : (null)
   +0x098 DdagNode         : 0x0000014f`889a3680 _LDR_DDAG_NODE
   +0x0a0 NodeModuleLink   : _LIST_ENTRY [ 0x0000014f`889a3680 - 0x0000014f`889a3680 ]
   +0x0b0 LoadContext      : 0x0000014f`889f6e20 _LDRP_LOAD_CONTEXT
   +0x0b8 ParentDllBase    : 0x00007ff9`34990000 Void
```

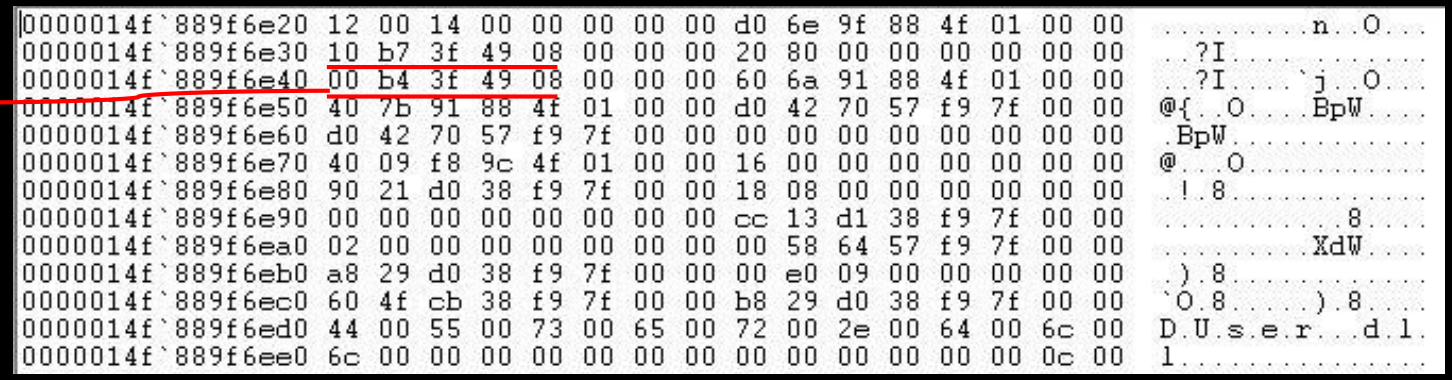
_LDR_DDAG_NODE

```
0:018> dt _LDR_DDAG_NODE 14f889a3680
Windows_UI!_LDR_DDAG_NODE
   +0x000 Modules          : _LIST_ENTRY [ 0x0000014f`88917be0 - 0x0000014f`88917be0 ]
   +0x010 ServiceTagList   : (null)
   +0x018 LoadCount        : 1
   +0x01c LoadWhileUnloadingCount : 0
   +0x020 LowestLink       : 0
   +0x028 Dependencies     : _LDRP_CSLIST
   +0x030 IncomingDependencies : _LDRP_CSLIST
   +0x038 State            : 4 ( LdrModulesSnapping )
   +0x040 CondenseLink     : _SINGLE_LIST_ENTRY
   +0x048 PreorderNumber   : 0
```

```
enum State {
LdrModulesPlaceHolder = 0,
LdrModulesMapping,
LdrModulesMapped,
LdrModulesWaitingForDependencies,
LdrModulesSnapping,
LdrModulesSnapped,
LdrModulesCondensed,
LdrModulesReadyToInit,
LdrModulesInitializing
};
```

Leak Stack Address From LdrpWork Structure

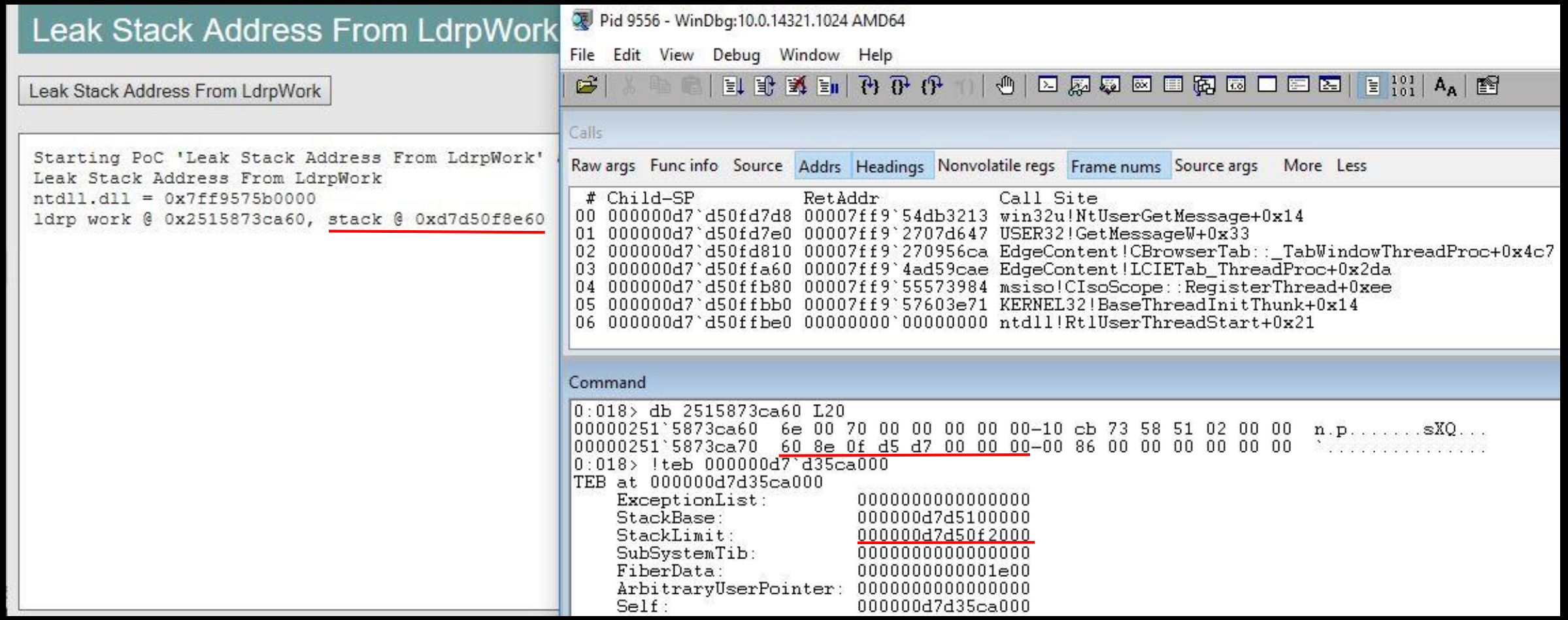
stack address



```
0000014f`889f6e20 12 00 14 00 00 00 00 00 d0 6e 9f 88 4f 01 00 00  .........n..O...
0000014f`889f6e30 10 b7 3f 49 08 00 00 00 20 80 00 00 00 00 00 00  ..?I.... .......
0000014f`889f6e40 00 b4 3f 49 08 00 00 00 60 6a 91 88 4f 01 00 00  ..?I....`j..O...
0000014f`889f6e50 40 7b 91 88 4f 01 00 00 d0 42 70 57 f9 7f 00 00  @{..O....BpW....
0000014f`889f6e60 d0 42 70 57 f9 7f 00 00 00 00 00 00 00 00 00 00  .BpW............
0000014f`889f6e70 40 09 f8 9c 4f 01 00 00 16 00 00 00 00 00 00 00  @...O...........
0000014f`889f6e80 90 21 d0 38 f9 7f 00 00 18 08 00 00 00 00 00 00  .!.8............
0000014f`889f6e90 00 00 00 00 00 00 00 00 cc 13 d1 38 f9 7f 00 00  ...........8....
0000014f`889f6ea0 02 00 00 00 00 00 00 00 00 58 64 57 f9 7f 00 00  .........XdW....
0000014f`889f6eb0 a8 29 d0 38 f9 7f 00 00 00 e0 09 00 00 00 00 00  .).8............
0000014f`889f6ec0 60 4f cb 38 f9 7f 00 00 b8 29 d0 38 f9 7f 00 00  `O.8.....).8....
0000014f`889f6ed0 44 00 55 00 73 00 65 00 72 00 2e 00 64 00 6c 00  D.U.s.e.r...d.l.
0000014f`889f6ee0 6c 00 00 00 00 00 00 00 00 00 00 00 00 00 0c 00  l...............
```

- Two stack addresses are saved in LdrpWork structure, and they belong to the thread on which the loading of module operates.
- LdrpWork structure stays in LdrpWorkQueue only for a very short time, and it will soon be removed from the queue for processing, which makes it impossible to trace it by reading it from the queue.
- The size of LdrpWork structure (PlaceHolder) is variable due to the difference in length of module name, so its location (such as LFH bucket) in LdrpHeap can’t be accurately predicted.
- But what if we can force the LdrpWork to be allocated in an expected size? ☺ By combining data-only attack technique with the knowledge about Windows 10 segment Heap Fengshui, it will not be difficult for us to reliably locate one LdrpWork instance in memory and leak the stack address out of it.

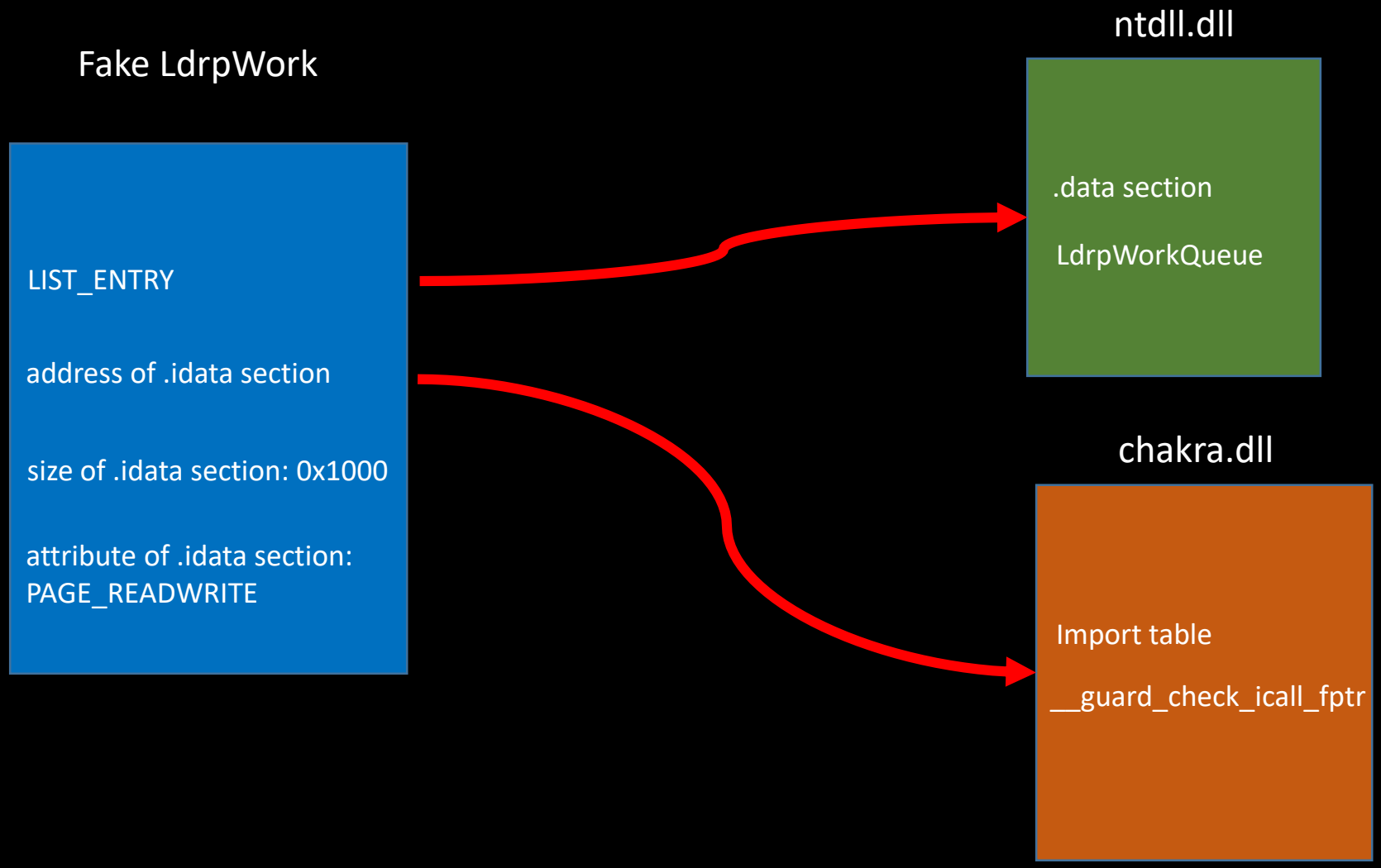
Leak Stack Address from LdrpWork Structure



CFG Bypass: Abuse LdrpWork Mechanism

- No protection on LdrpWork doubly-linked list (ntdll!LdrpWorkQueue) to prevent it from being modified (such as inserting a new node).
- LdrModuleSnapping work can be leveraged to change the memory attribute of arbitrary address. ntdll!LdrpDoPostSnapWork fails to enforce a check on the target address to make sure it indeed belongs to .idata section of certain module that is in LdrModuleSnapping state.
- The whole exploitation process is virtually as simple as manipulating a linked list: first fake a LdrpWork structure (LdrModuleSnapping work) that carries all the information needed for the memory attribute change, then insert it to doubly-linked list and wait for it to be processed. Normally this won’t take long as LdrpDrainWorkQueue operation will be triggered at many places. Please note that the data in the fake LdrpWork must be correctly populated to ensure it can eventually reach ntdll!LdrpDoPostSnapWork.

The Diagram of CFG Bypass Scheme



CFG Bypass: Abuse LdrpWork Mechanism

```
ntdll!LdrpDoPostSnapWork:
00007ff9`575fc30c 48895c2410      mov     qword ptr [rsp+10h],rbx
00007ff9`575fc311 57              push    rdi
00007ff9`575fc312 4883ec30        sub     rsp,30h
00007ff9`575fc316 488bd9          mov     rbx,rcx
00007ff9`575fc319 33ff            xor     edi,edi
00007ff9`575fc31b 8bcf            mov     ecx,edi
00007ff9`575fc31d 488d5360        lea     rdx,[rbx+60h]
00007ff9`575fc321 48393a          cmp     qword ptr [rdx],rdi
00007ff9`575fc324 7424            je      ntdll!LdrpDoPostSnapWork+0x3e (00007ff9`575fc34a)
00007ff9`575fc326 448b8b80000000  mov     r9d,dword ptr [rbx+80h]
00007ff9`575fc32d 488d442440      lea     rax,[rsp+40h]
00007ff9`575fc332 4c8d4368        lea     r8,[rbx+68h]
00007ff9`575fc336 4889442420      mov     qword ptr [rsp+20h],rax
00007ff9`575fc33b 4883c9ff        or      rcx,0FFFFFFFFFFFFFFFFh
00007ff9`575fc33f e86ca20500      call    ntdll!NtProtectVirtualMemory (00007ff9`576565b0)
```

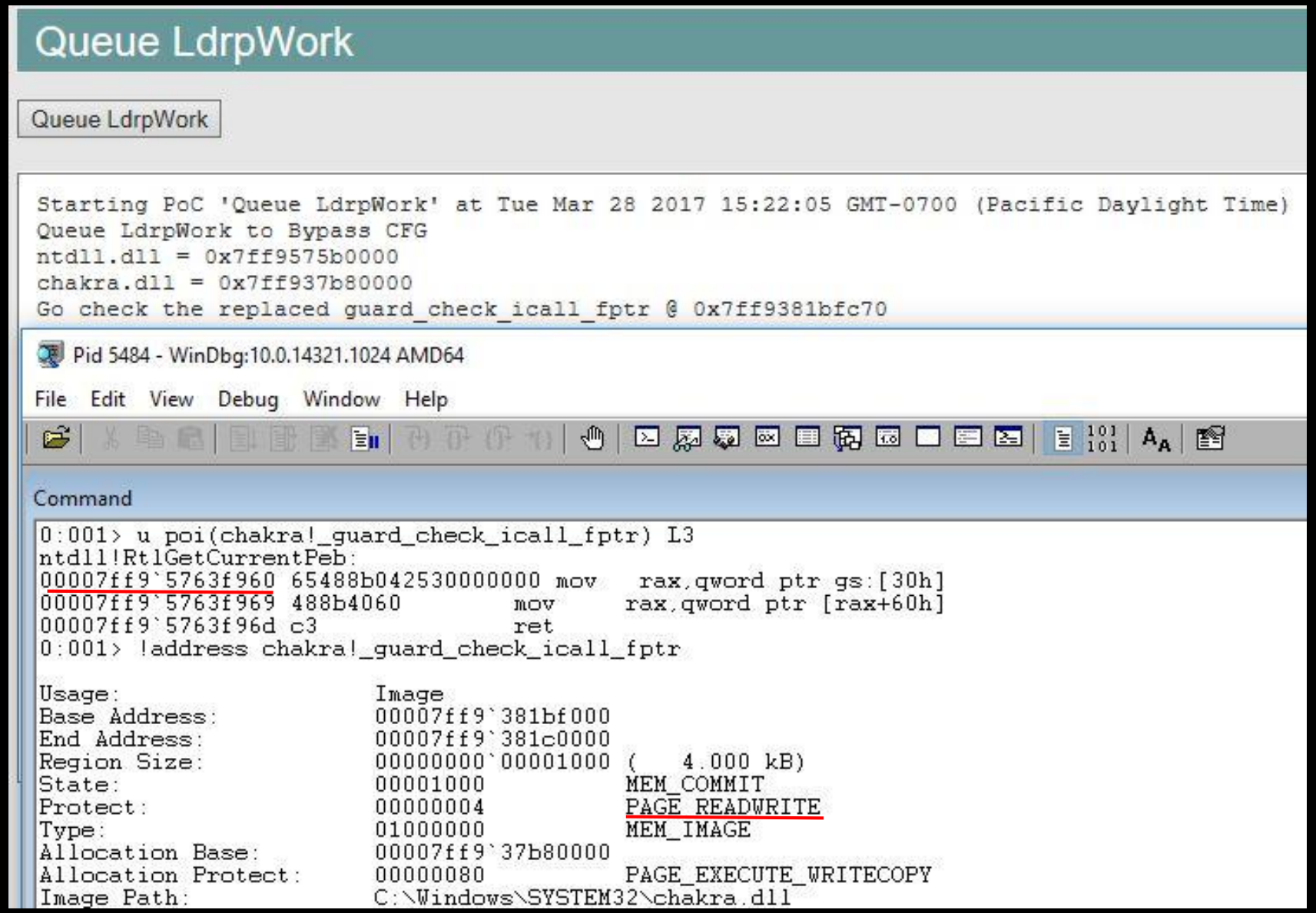
Command

```
0:017> r
rax=000000b715ff7ea0 rbx=0000027d2d455730 rcx=ffffffffffffffff
rdx=0000027d2d455790 rsi=0000000000000000 rdi=0000000000000000
rip=00007ff9575fc33f rsp=000000b715ff7e60 rbp=000000b715ff8200
 r8=0000027d2d455798  r9=0000000000000004 r10=00000000000014a6
r11=000000000000006c r12=00007ff9577042d0 r13=0000000000000004
r14=0000027d2d4556b8 r15=0000000000000000
iopl=0         nv up ei ng nz na po nc
cs=0033  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000286
ntdll!LdrpDoPostSnapWork+0x33:
00007ff9`575fc33f e86ca20500      call    ntdll!NtProtectVirtualMemory (00007ff9`576565b0)
0:017> ? poi(27d2d455790)
Evaluate expression: 140708364942448 = 00007ff9`381bfc70
0:017> ln 7ff9`381bfc70
Browse module
Set bu breakpoint

(00007ff9`381bfc70)   chakra!_guard_check_icall_fptr   |  (00007ff9`381bfc78)   chakra!_guard_dispatch_icall_fptr
```

In this PoC, we choose to change the memory attribute of chakra!_guard_check_icall_fptr

CFG Bypass: Abuse LdrpWork Mechanism



Solution for CFG Bypass Abusing LdrpWork

Bypass	Status
Non-enlightened Just-in-Time (JIT) compilers	Mitigated in latest version of Edge on Windows 10 (Chakra, Adobe Flash, and WARP)
Multiple non-instrumented indirect calls reported to our Mitigation Bypass Bounty	Mitigated in latest version of Edge on Windows 10
Calling sensitive APIs out of context	NtContinue/longjmp – mitigated for all CFG enabled apps on Windows 10
	VirtualProtect/VirtualAlloc – mitigated in latest version of Microsoft Edge on Windows 10
	LoadLibrary – mitigated in latest version of Microsoft Edge on Windows 10 via CIG
	WinExec – mitigated in Edge on Windows 10 anniversary edition via child process policy
Corrupting mutable read-only memory	Known limitation that we are exploring solutions for

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 part II of our data-only attack series. In our future conference talks, we are looking to present the advanced data-only attack techniques (data-only combined with other exploitation techniques). Stay tuned!

- You are welcomed to send questions to
  - Bing Sun @ bing_sun@mcafee.com
  - Chong Xu @ chong_xu@mcafee.com

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

- Special thanks to Stanley Zhu, 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