

原文链接：<https://www.zerodayinitiative.com/blog/2018/12/17/seeing-double-exploiting-a-blind-spot-in-memgc>

这是我们评选的2018年五大有趣案例的第一个。这些评选出来的bug都具有一些独特的元素，使得其与今年发布的大约1400条报告不同。我们首先来看Pwn2Own冠军的——

在2018年Pwn2Own大会上，Richard Zhu([fluorescence](#))成功攻陷多个目标，获得了世界破解大师（[Master of PWN](#)）的称号。他攻陷的目标之一是Microsoft Edge，使用的利用链包括两个Use-After-Free (UAF)漏洞。其中一个UAF漏洞十分引人注目，以至于被列为我们今年的五大漏洞之一，将会在本系列博客中对此进行详细介绍。这个漏洞编号是[CVE-2018-8179](#)

我们来深入研究这个漏洞的一些PoC代码，看看是什么让它如此惊人：

```
arr1 = [];  
for (i = 0; i < 0x800000; i++){  
    arr1.push([i]);  
}  
  
arr2 = new Array(2);  
arr2[0] = arr1[0x7e0000];  
  
arr2.__defineGetter__(1, function(){  
    delete arr2[0];  
    arr1.length = 0;  
    arr1 = [];  
    for (let i=0;i<0xa000;i++){  
        let ua = new Uint32Array(0x8000/4);  
        for (let j=0;j<ua.length;j++) {  
            ua[j] = 0x41414140+j%8; }  
        arr1.push(ua);  
    }  
    return 0;  
});  
  
var iceTransport = new RTCIceTransport();  
iceTransport.setRemoteCandidates(arr2);
```

(1) This just creates a large number of objects in memory

(2) arr2[0] now refers to one of the objects created above

(3) This will iterate over arr2. See text.

(4) When accessing arr2[1], remove all references to all objects created above, including arr2[0]

(5) Apply memory pressure, causing the original arr2[0] object to be garbage collected and reclaimed

图片显示了poc代码和一些指示操作顺序的注释。主要操作是从步骤3中setRemoteCandidates的调用开始的，这个API需要传入一个JavaScript数组。如图中所示，在遍历

为了实现这幅图中的内容，我们需要更多地了解setremotecandidate处理其参数时发生的事情，它做了如下操作：

1. 创建一个名为CModernArray<>的内部数组结构。
2. 遍历arr2。对于每个元素，获取一个指向该元素的指针，并将其添加到CModernArray<>中。
3. 迭代CModernArray<>，依次处理每个JavaScript对象。

CModernArray<>是在edgehtml.dll中定义的一个c++类。至关重要，它将数据存储在与MemGC堆分配的缓冲区中。概括总结下POC的操作：edgehtml.dll在arr2上

把所有这些信息汇总在一起后，我们现在可以理解这里存在一个漏洞是多么不可思议。在整个过程中，所有涉及的对象(Javascript数组等)都被分配到MemGC堆上。此外，为什么会这样呢？

我现在要告诉你一个可怕的秘密。

并没有这样一个无所不知的“MemGC堆”

实际上有两个MemGC堆，它们对彼此的分配是不可见的。

这两个MemGC堆如下所示，其中一个MemGC堆会在浏览器的JavaScript引擎Chakra中内部使用，所有基于堆的JavaScript对象以及许多内部Chakra数据结构都存储在这个Explorer在2014年7月首次引入的内存保护机制。DOM堆用于所有DOM对象，以及从edgehtml.dll执行的大多数其他堆分配。

这两个堆共享一个实现，但是它们由chakra!Memory::Recycler类的两个不同实例表示。当垃圾回收发生时，在“标记”阶段，回收程序会扫描所有存活的堆分配，以及堆栈和堆内存。顺便提一下，甚至有存在两个以上的MemGC堆。JavaScript执行的每个线程都有自己的Chakra堆实例。通常这不会造成问题，因为JavaScript对象不会与创建它的线程以外。现在我们可以理解在运行图中poc代码时发生了什么，在步骤5(见图)中，内存压力迫使Chakra堆进行垃圾收集。这是由与Chakra堆相关的Recycler实例执行的。在扫描堆栈时，

总结

我们已经证明，“MemGC是无所不知的”这种概念是一种误解。虽然MemGC作为一种缓解措施非常成功，但它并非完全没有缺点。这对于考虑如何打补丁也是有指导意义的。现在，在将每个对象添加到CModernArray<>之前，edgehtml调用chakra::JsVarAddRef来显式地将对象固定在内存中。至于edgehtml，你可以关注我的Twitter@HexKitchen,或者关注我们的团队以了解最新的漏洞利用技术和安全补丁。请继续关注下一个年度五大漏洞相关博客，它将于明天发布。

点击收藏 | 0 关注 | 1

[上一篇：使用RouterSploit控制路由器](#) [下一篇：Phar的一些利用姿势](#)

1. 0 条回复
- 动动手指，沙发就是你的了！

[登录](#) 后跟帖

先知社区

[现在登录](#)

热门节点

[技术文章](#)

[社区小黑板](#)

目录

[RSS](#) [关于社区](#) [友情链接](#) [社区小黑板](#)