CVE-2019-13720：Chrome 0-day 漏洞利用

原创 ang010ela 嘶吼专业版 2019-11-07

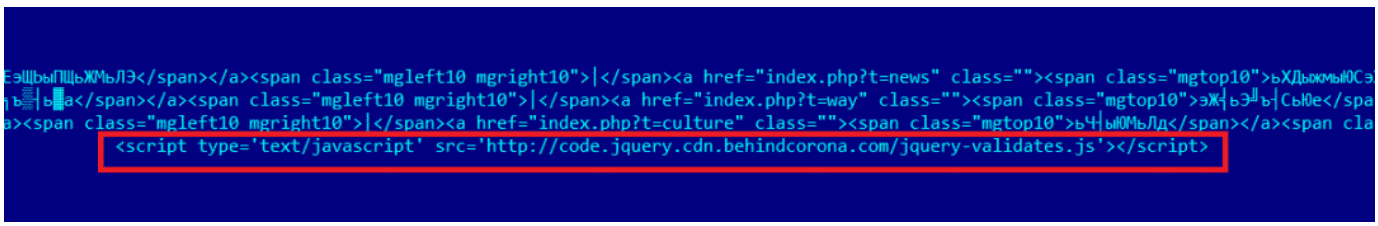
摘要

Kaspersky研究人员近日发现一个Google Chrome浏览器的新的未知漏洞利用。经Google研究人员确认，是一个0 day漏洞，CVE编号为CVE-2019-13720。

研究人员将相关攻击活动命名为Operation WizardOpium。目前还无法将该攻击活动与已知的攻击者联系在一起。但研究人员发现部分代码与Lazarus攻击中的代码很相似。从被攻击的站点来看，与早期DarkHotel攻击活动类似。

技术细节

攻击活动使用了一种对韩语新闻门户的水坑形式的注入。攻击者会把恶意JS代码插入到主页中，恶意JS代码会从远程站点加载一个分析脚本。



重定向到漏洞利用加载页面

一个主页index页面会从hxxp://code.jquery.cdn.behindcorona[.]com/处加载一个标记为JS tag的远程脚本。

该脚本会加载一个名为.charlie.XXXXXXXX.js的脚本。JS代码会通过比较浏览器的用户代理来检查受害者的系统是否感染，用户代理会在64位的Windows版本上运行，而非WOW64进程，并尝试获取浏览器的用户名和版本。

该漏洞会尝试利用Google Chrome浏览器的bug，脚本会检查浏览器版本是否大于65（当前Chrome版本为78）：

```
if (navigator.userAgent.indexOf("Win64") == -1 || navigator.userAgent.indexOf("WOW64") != -1)
        return ;

if (navigator.userAgent.indexOf("Windows NT 6.1") == -1)
        return ;

let r = navigator.userAgent.indexOf("Chrome/");

if (r == -1)
        return ;

if (parseInt(navigator.userAgent.substr(r + "Chrome/".length, 3)) < 65)
        return ;
```

profiling 脚本(.charlie.XXXXXXXX.js)中的chrome版本检查

在浏览器版本检查后，脚本会执行一些到攻击者控制的服务器（behindcorona[.]com）的AJAX请求，服务器中的路径名会指向传递给脚本(xxxxxxx.php)的参数。第一个请求对于获取一些重要信息是非常必要的。这些信息包括一些十六进制编码的字符串来告诉脚本那些真实漏洞利用代码块会从服务器下载，一些到图像文件的URL，图像文件嵌入了到final payload的key和解密漏洞利用代码的部分块。

404	HTTP	behindcorona.com	/favicon.ico
200	HTTP	behindcorona.com	
200	HTTP	behindcorona.com	
200	HTTP	behindcorona.com	
200	HTTP	behindcorona.com	
200	HTTP	behindcorona.com	
200	HTTP	behindcorona.com	
200	HTTP	behindcorona.com	
200	HTTP	behindcorona.com	
200	HTTP	behindcorona.com	
200	HTTP	behindcorona.com	

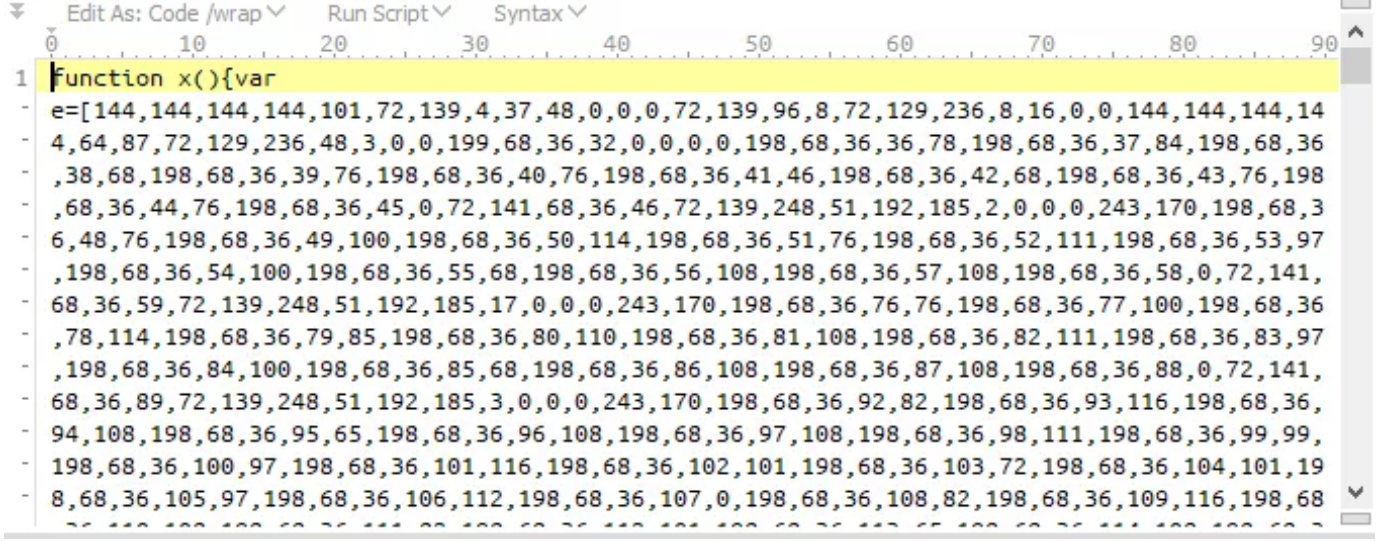
漏洞利用链–到xxxxxxx.php的AJAX请求

下载了所有块后，RC4脚本会解密和连接所有的块，这样攻击者就有了含有所有浏览器漏洞利用的新的JS代码。为了解密这些部分，需要使用之前提取的RC4 key。

```
try {
    var n = navigator.userAgent.split("Chrome/")[1].split(" Safari/")[0];
    n = parseInt(n.substr(0, 2));
    if (n != 77 && n != 76) {
        return
    }
```

版本检查

浏览器漏洞利用脚本是经过混淆的，在反混淆后，研究人员发现了针对用户代理字符串的另一个检查，这次检查的浏览器版本是76或77。也就是说漏洞利用开发者只在这些版本上工作，或其他漏洞利用已经用于之前的Chrome版本中了。



混淆的漏洞利用代码

在浏览器内置的BigInt 类中，有一些函数运行的函数。BigInt 类的作用是在JS代码中进行64位的算术运算。漏洞利用开发者通过与32位数来进行实现这些函数。BigInt 类的作用在浏览器代码中实现的速度更快。漏洞利用开发者并不都使用64位，而是在小范围的数上进行运算。

```
function O(e) {
    var t = D(e);
    var n = (BigInt(1) << BigInt(21)) - BigInt(1);
    var r = (e & n) >> BigInt(14);
    var i = t + (BigInt(4096) + r * BigInt(32));
    return i
}

function k(e, t) {
    var n = D(e);
    var r = t << BigInt(14);
    return n + r
}

function E(e) {
    var t = D(e);
    var n = (BigInt(1) << BigInt(21)) - BigInt(1);
    var r = (e & n) >> BigInt(14);
    return r
}
```

对64位数字进行处理的代码段

在真实代码中有很多的函数和变量并没有使用。也就是说这些代码应该是用来调试的，或还没有用于生产中。

代码的大部分使用了一些有特定浏览器组件相关的类。因为该漏洞还没有被修复，本文为未介绍该有漏洞的组件的细节。

该漏洞利用使用了2个线程之间的一个竞争条件漏洞，漏洞的原因是由于错失了适当的同步。漏洞给了攻击者一个UAF条件，因为UAF会导致漏洞执行的场景。

漏洞利用首先会尝试触发UAF来执行关于重要64位地址的信息泄露。这会引发：

1、如果地址成功泄露，就说明漏洞利用正常工作。

2、泄露的地址会用来了解堆或栈的地址，用来处理ASLR技术。

3、提供过搜索附近地址来寻找其他漏洞利用的有用指针。

然后会尝试用递归函数来创建一个大对象的部分。这是通过一些确定性的堆布局来实现的，这对成功的漏洞利用来说是非常重要的。同时会尝试使用堆喷射技术来使用之前UAF部分的指针。该技术会用来引发混淆，使攻击者可以在两个位于相同内存区域的不同的对象之间操作。

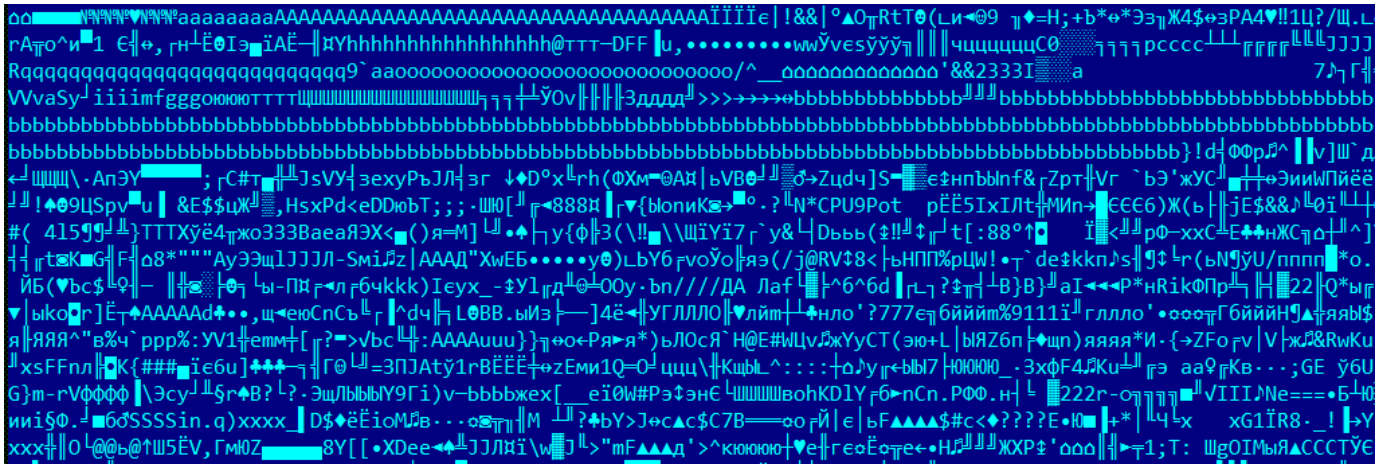
漏洞利用会尝试执行大量的操作来分配或释放内存，以给攻击者任意的读写原语。这是用来伪造特殊的对象的，该对象可以与WebAssembly 和FileReader一起来执行嵌入shellcode payload的代码执行。

```
00000000: 90909090                nop
00000004: 6548                    dec         eax
00000006: 8B042530000000          mov         eax,[000000030]
0000000D: 48                      dec         eax
0000000E: 8B6008                  mov         esp,[eax][8]
00000011: 48                      dec         eax
00000012: 81EC08100000            sub         esp,000001008
00000018: 90909090                nop
0000001C: 40                      inc         eax
0000001D: 57                      push        edi
0000001E: 48                      dec         eax
0000001F: 81EC30030000            sub         esp,000000330 ;'  ♥0'
00000025: C744242000000000        mov         d,[esp][020],0
0000002D: C64424244E              mov         b,[esp][024],04E ;'N'
00000032: C644242554              mov         b,[esp][025],054 ;'T'
00000037: C644242644              mov         b,[esp][026],044 ;'D'
0000003C: C64424274C              mov         b,[esp][027],04C ;'L'
00000041: C64424284C              mov         b,[esp][028],04C ;'L'
00000046: C64424292E              mov         b,[esp][029],02E ;'.'
0000004B: C644242A44              mov         b,[esp][02A],044 ;'D'
00000050: C644242B4C              mov         b,[esp][02B],04C ;'L'
00000055: C644242C4C              mov         b,[esp][02C],04C ;'L'
0000005A: C644242D00              mov         b,[esp][02D],0
0000005F: 48                      dec         eax
```

第一阶段shellcode

Payload分析

Final payload是以加密的二进制文件(worst.jpg)下载的，同时也是shellcode解密的。



加密的payload – worst.jpg

解密后，恶意软件模块会以updata.exe的形式释放到磁盘中并执行。为了实现驻留，恶意软件会在Windows任务计划器中安装任务。

Payload installer是一个RAR SFX文件，含有以下信息：

- 文件大小: 293,403
- MD5: 8f3cd9299b2f241daf1f5057ba0b9054
- SHA256: 35373d07c2e408838812ff210aa28d90e97e38f2d0132a86085b0d54256cc1cd

压缩文件中含有2个文件：

```
;The comment below contains SFX script commands
        Path=%programdata%
        Setup=iohelper.exe
        Silent=1
        Overwrite=1
        Testing         msdisp64.exe                    OK
        Testing         iohelper.exe                    OK
        All OK
```

- 文件名: iohelper.exe

MD5: 27e941683d09a7405a9e806cc7d156c9
SHA256: 8fb2558765cf648305493e1dfea7a2b26f4fc8f44ff72c95e9165a904a9a6a48

- 文件名: msdisp64.exe

MD5: f614909fbd57ece81d00b01958338ec2

SHA256: cafe8f704095b1f5e0a885f75b1b41a7395a1c62fd893ef44348f9702b3a0deb

这两个文件的编译时间是相同的，时间戳为 2019年10月8日01:49:31。
主模块msdisp64.exe会尝试从硬编码的C2服务器集下载下一阶段。下一阶段位于C2服务器的文件夹名为受害者计算机名的文件夹中，所以攻击者就有了哪些计算机受感染了，并将下一阶段模块放在C2服务器的特定文件夹中。

注：本文翻译自：https://securelist.com/chrome-0-day-exploit-cve-2019-13720-used-in-operation-wizardopium/94866/

END