IIS 6.0 WebDAV远程代码执行漏洞分析—【CVE-2017-7269】

菠菜 / 2017-03-30 15:04:00 / 浏览数 7071 技术文章 技术文章 顶(0) 踩(0)

漏洞描述：

3月27日，在Windows 2003 R2上使用IIS 6.0
爆出了0Day漏洞（CVE-2017-7269），漏洞利用PoC开始流传，但糟糕的是这产品已经停止更新了。网上流传的poc下载链接如下。

github地址：https://github.com/edwardz246003/IIS_exploit

结合上面的POC，我们对漏洞的成因及利用过程进行了详细的分析。在分析过程中，对poc的exploit利用技巧感到惊叹，多次使用同一个漏洞函数触发，而同一个漏洞同一

调试环境：

虚拟机中安装Windows Server 2003企业版，安装iss6.0后，设置允许WebDAV扩展。使用的调试器为：windbg:6.7.0005.1

远程代码执行效果如下：

由上图可到，漏洞利用成功后可以network services权限执行任意代码。

漏洞分析：

漏洞函数

漏洞位于ScStoragePathFromUrl函数中,通过代码可以看到，在函数尾部调用memcpy函数时，对于拷贝的目的地址来自于函数的参数，而函数的参数为上层函数的局部变

通过伪代码更容易看出：

漏洞利用：

在POC中，可以看到发送的header中包含两部分<>标签，这会使上面的每个循环体都会运行两次，为了下面的描述方便，我们对这两个header的标签部分分别定义为HEA

漏洞利用流程：

在HrCheckIfHeader函数中，通过使用HEAD_A溢出，使用HEAD_B被分配到堆空间地址中。

在HrGetLockIdForPath函数中，再次通过使用HEAD_A溢出，使HEAD_B所在的堆地址赋值给局部对象的虚表指针，在该对象在调用函数时，控制EIP。

最终调用IEcb类的对象偏移0x24处的函数指针，控制EIP

漏洞利用主要在于HrCheckIfHeader函数与函数HrGetLockIdForPath中。

函数HrCheckIfHeader主要功能是对用户传递来的Header头进行有效性的判定。在函数中HrCheckIfHeader通过了while循环来遍历用户输入的Header头中的数据。

HrGetLockIdForPath主要功能是对传递来的路径信息进行加锁操作。在HrGetLockIdForPath函数中，也是通过while循环来遍历路径信息，同样也对应着两次调用漏洞函数

调试过程：

两次溢出控制EIP

对这4个调用漏洞函数的地方分别下断：

```
bp httpext!CParseLockTokenHeader::HrGetLockIdForPath+0x114 ".echo HrGetLockIdForPath_FIRST";
bp httpext!CParseLockTokenHeader::HrGetLockIdForPath+0x14f ".echo HrGetLockIdForPath_SECOND";
bp httpext!HrCheckIfHeader+0x11f ".echo HrCheckIfHeader_FIRST";
bp httpext!HrCheckIfHeader+0x159 ".echo HrCheckIfHeader_SECOND";
```

调试程序，共会断下6次，我们对这6次断点处漏洞函数在利用时的功能进行归纳：

第一次：

暂停在HrCheckIfHeader _FIRST，对漏洞利用没有影响

第二次：

断在HrCheckIfHeader
_SECOND，此处调用漏洞函数的目的是为了使用HEAD_A标签，来溢出漏洞函数，目的是使用HEAD_A标签中的堆地址覆盖栈中的地址，此堆地址会在随后使用。

运行漏洞函数前，

运行过漏洞函数后，可以看到栈空间中的0108f90c位置处的内容已经被覆盖成了680312c0，680312c0正是一个堆中的地址。

第三次：

暂停在HrCheckIfHeader_FIRST,此时漏洞函数的作用是，将HEAD_B标签拷贝到上面的堆地址中。本来正常的程序在这里会将用户传递进来的HEADER拷贝到栈空间中，但

第四次：

暂停到HrCheckIfHeader_FIRST，对漏洞利用没有影响

第五次：

HrCheckIfHeader_SECOND，此处调用漏洞函数的目的是为了使用HEAD_A标签，来溢出漏洞函数，目的是使用HEAD_A标签中的堆地址覆盖栈中的地址，此堆地址会在随
ebp-14 将栈中的地址改成了与堆中的地址
680312c0，在这里ebp-14的地址也被覆盖，这个地址在下面第六次的溢出时，会赋值给对象指针，在这里就控制了ebp_14的值，也就可以控制下一步中的对象指针。

第六次：

HrCheckIfHeader_FIRST在这个函数下面的子子函数中会调用虚函数，从而控制EIP。

总结一下，在上面六次调用处，需要关注的利用过程是：

1）
第二次与第三次处是必须的，因为没有第二次处的利用，就不会有第三次处的把HEAD_B拷贝到堆中。没有堆中的地址在第六次调用时就没法控制虚表指针。所以没有第二次

2）
第五次再次把栈溢出，把堆的地址写到了局部变量中，才导致第六次能成功调用虚函数。因为第六次调用虚函数时，是调用的局部变量的虚函数。如果没有第五次断点处的溢

由此可以看出，两次对漏洞函数溢出操作，其中一次溢出操作(第二次断点处)将栈地址改写为堆地址，保证了HEAD_B被写入到堆中，另外一次溢出操作(第五次断点处)将局部

ROP

控制EIP后，使用ROP技术绕过GS的保护。

使用SharedUserData的方法执行自定义的函数

来到shellcode处：

Shellcode进行一次循环解码：

解码完成后，就是长得比较漂亮的shellcode了

缓解方案：

l 禁用 IIS 的 WebDAV 服务

l 使用 WAF相关防护设备

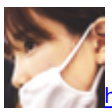
l 建议用户升级到最新系统 Windows Server 2016。

总结

通过分析可以看到，漏洞原理只是因为没有对拷贝函数的长度做判断，而导致了栈溢出。这也提醒广大程序员们，慎用不安全的内存操作函数，在编译代码时开启所有保护。

上一篇：0CTF2017 EasiestP... 下一篇：[先知公告] 先知众测2017年2...

1. 5 条回复


hades 2017-03-30 15:39:08

沙发必须的我的 好辛苦的

0 回复Ta


hades 2017-03-31 02:12:32

https://github.com/dmchell/metasploit-framework/pull/1/commits/9e8ec532a260b1a3f03abd09efcc44c30e4491c2

Msf 添加模块
vi iis6.rb //新建rb文件
mv iis6.rb /usr/share/metasploit-framework/modules/exploits/ //移动到msf目录
msfconsole //启动msf
reload_all //重载所有模块
use exploits/iis6 //加装Exp
配置Msfconsole

0 回复Ta


cryin 2017-03-31 02:19:01

赞，看了好几篇，，唯有这篇对漏洞原理及利用将的最详细！！

0 回复Ta


k0shl 2017-04-01 15:15:45

文中有一处错误，ROP并非用来bypass GS，而是用来利用shared user data绕过DEP，详情可以看TK在CanSecWest 2013中关于shared user data的技巧。bypass GS在第一次栈溢出的时候就已经完成了，其实栈里还有很多其他有意思的结构体可以利用，并非只能利用ebp - 328栈位置这一处变量。

0 回复Ta

steven1881 2017-10-27 06:50:05

借鉴复现漏洞

0 回复Ta

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