

原文：<http://www.offensiveops.io/tools/cobalt-strike-bypassing-windows-defender-with-obfuscation/>

对于所有红队队员来说，在投递有效载荷的时候，总要面临绕过安全软件的检测的挑战。而且，就像其他安全解决方案一样，Windows Defender在检测Cobalt Strike等工具生成的普通有效载荷方面，也变得越来越驾轻就熟了。

在本文中，我们将以Cobalt Strike生成的PowerShell载荷为例，讲解如何通过混淆处理，使其绕过Windows 10 系统上的Windows Defender。在绕过Windows Defender方面，虽然该方法算不上是最优雅或最简单的，但是在工作中我们一直都在使用这种方法，并且一直很有效。

创建有效载荷的过程如下所示：

这时，会生成一个包含PowerShell命令的payload.txt文件。

但是，如果直接将上述文件放到受害者机器上运行的话，就会被Windows Defender检测到安全威胁。

为了绕过Windows Defender，首先需要了解一下Cobalt Strike生成有效载荷的机制，然后修改相应的签名，从而使Windows Defender误认为它是安全的。

首先，不难看出，这些有效载荷命令是base64编码的，我们通过格式就能看出来；此外，这也可以通过PowerShell的-encodedcommand选项检测出来。

为了对命令进行解码，我们需要通过下面的命令

```
powershell.exe -nop -w hidden -encodedcommand
```

删除一部分字符串，并保留其余部分。

然后，使用以下命令对剩余字符串进行解码。

```
echo 'base64 payload' | base64 -d
```

上述命令解码后的字符串，实际上还是base64编码字符串，但尝试解码时，发现该方法无效了，因为得到的都是乱码——从PowerShell命令中的 IEX (New-Object IO.StreamReader(New-Object IO.Compression.GzipStream(\$s[IO.Compression.CompressionMode]::Decompress))).ReadToEnd()部分来看，该字符串好像还经过了Gzip压缩处理。

现在，我们需要了解这个命令的具体内容，因为这是实际触发Windows Defender的部分，即有效载荷部分。利用谷歌搜索，我发现一个正好可以用来解决这个问题的脚本，地址为：
<http://chernodv.blogspot.com.cy/2014/12/powershell-compression-decompression.html>

```
$data = [System.Convert]::FromBase64String('gzip base64')
$ms = New-Object System.IO.MemoryStream
$ms.Write($data, 0, $data.Length)
$ms.Seek(0,0) | Out-Null
$sr = New-Object System.IO.StreamReader(New-Object System.IO.Compression.GZipStream($ms, [System.IO.Compression.CompressionMode]::Decompress))
$sr.ReadToEnd() | set-clipboard
```

该脚本首先对base64字符串进行解码，然后进行解压，这样就能得到相应的代码。此外，它还会将输出的内容复制到剪贴板，以便将其粘贴到稍后用到的文本文件中。

\$var_code变量保存的就是Windows Defender正在检测的有效载荷，所以，为了绕过它，我们需要对这个变量做些手脚。

如果进一步对\$var_code进行解码的话，会得到一串ASCII字符，但现在还无需进行彻底解码。

```
$enc=[System.Convert]::FromBase64String('encoded string')
```

现在，我们来读取部分内容，具体命令如下所示：

```
$readString=[System.Text.Encoding]::ASCII.GetString($enc)
```

上图展示了与用户代理和攻击者IP有关的一些信息。

下面，我们要做的是读取相关有效载荷并进行混淆处理，以绕过Windows Defender的检测。进行这项工作的时候，最佳的工具恐怕就是[Daniel Bohannon](#)提供的Invoke-Obfuscation了。该项目的Github页面可以在[这里](#)找到。

现在启动Invoke-Obfuscation，具体命令如下所示：

```
Import-Module .\Invoke-Obfuscation.psdl
Invoke-Obfuscation
```

现在，有效载荷的哪些部分需要进行混淆，我们必须规定好，具体可以通过以下命令完成

```
Set scriptblock 'final_base64payload'
```

该工具将读取脚本代码，然后询问接下来的处理方式。

在本例中，我先选择了COMPRESS，然后又选择1。当然，这并不意味着其他选项将无法正常工作，但是至少在编写本文时，我的选择是能够绕过Windows Defender的。然后，Invoke-Obfuscation会根据我们的选择进行相应的处理，并输出可以绕过Windows Defender的PowerShell命令。

然后，只需键入Out和PowerShell脚本的保存路径即可。

```
Out c:\payload.ps1
```

这样，就会得到经过压缩的有效载荷，具体如下所示。

实际上，我们最终的目的，就是用Invoke-Obfuscation新建的有效载荷替换[Byte[]]\$var_code = [System.Convert]::FromBase64String。为此，我定义了一个新的变量，我称之为\$evil，只是用于存放Invoke-Obfuscation的输出内容。

注意，对于Invoke-Obfuscation的输出结果，最后一个|之后部分是应该删除掉的，因为它是用于执行命令的命令。但是，我们无需担心这个问题，因为Cobalt Strike模板会替我们删除这些内容。

将编辑后的脚本保存到PowerShell文件中，并执行该文件。如果您使用的是Cobalt Strike，将会看到一个beacon；如果您使用的是@sec_groundzero的[Aggressor Script](#)的候，会看到一个Slack通知。

如果我们用Process Hacker来观察vanilla CS有效载荷和修改后的CS有效载荷之间的变化的话，就会发现beacon的低层行为并没有发生任何变化。

点击收藏 | 3 关注 | 2

[上一篇：利用暴力攻击破解登陆密码](#) [下一篇：某 Java 急速开发框架分析与挖掘](#)

1. 2 条回复



[isec](#) 2018-03-18 13:04:25

感谢大佬分享

0 回复Ta



[mss****](#) 2018-03-18 15:34:29

感谢您的回复，翻译水平有限，还望多多指教。

[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](#)