How to Configure the Azure Connectivity Agent

https://campus.barracuda.com/doc/73718954/

> Only use the Azure connectivity agent, if you cannot use Cloud Integration for Azure or AWS. For more information, see Cloud Integration for AWS and Cloud Integration for Azure.

The Azure Connectivity Agent allows you to use the Barracuda CloudGen Firewall in the Azure Cloud as the default gateway for your Windows Server Azure VMs. The Connectivity Agent must be installed on all VMs in the VNET that initiate outgoing connections.

Before You Begin

- Deploy your Barracuda CloudGen Firewall in the Azure Cloud.
- Deploy your Windows Server VMs in the same subnet(s) as the CloudGen Firewall.
- Create a Folder on the Windows Server VM and add the following files:
  - A PFX client certificate for each Windows Server VM. For more information, see How to Create Certificates with XCA.
  - Create a PowerShell script file **AzureConnectivityAgent.ps1** and copy the deployment script below.
  - Barracuda Network Access Client version 3.6 HF104 x64 or later. To download the Network Access Client, go to the Barracuda Download Portal.
- Create a client-to-site VPN on the Barracuda CloudGen Firewall. Verify that it can be reached from the subnets where the Windows Server VMs are located. For more information, see Client-to-Site VPN.
- (optional) Lock down the private subnets with Azure Network Security Groups to make sure outgoing traffic is sent through only the firewall.

Example Azure Connectivity Deployment Script

Use the PowerShell script below to deploy the VPN client as a service on your Windows Server VM in Azure.

```
#
================================================================================
===
#
# Barracuda Azure Connectivity Agent Configuration and Setup Tool
# Copyright (C) 2014 Barracuda Networks, Inc.
#
```

```
# Version: 1.0.42
#
#
================================================================================
===

<#
.SYNOPSIS
Barracuda Azure Connectivity Agent Configuration and Setup Tool

.DESCRIPTION
Creates a customized configuration file (customer.inf) for silent
installation of the Barracuda Azure Connectivity Agent and/or installs the
software based on the given configuration file.
The parameters required to generate the configuration file can either be
specified directly by using the appropriate parameters or by answering the
respective questions at the command prompt.

.PARAMETER PfxFileName
The file name of the PFX container that contains an appropriate
certificate/private key (X.509) used for establishing a VPN connection.

.PARAMETER ConfigureOnly
Generates a new configuration file for silent installation, but does not
start installation of the Barracuda Azure Connectivity Agent software.
If neither -ConfigureOnly nor -InstallOnly is specified, the installation
process will be started immediately after a new configuration file has been
generated.

.PARAMETER InstallOnly
Installs the Barracuda Azure Connectivity Agent software by using the current
configuration file.
If neither -ConfigureOnly nor -InstallOnly is specified, the installation
process will be started immediately after a new configuration file has been
generated.

.PARAMETER PfxFilePassword
The password required to open the give PFX container file.

.PARAMETER ProfileDescription
The description of the VPN profile.

.PARAMETER ServerAddress
The IP address or hostname of the VPN server to which a VPN tunnel should be
established.
```

```
.PARAMETER TunnelMode
The Tunnel mode which should be used.
Possible values are: TCP, UDP, Hybrid

.PARAMETER EncryptionAlgorithm
The encryption algorithm which should be used for the VPN tunnel.
Possible values are: None, AES, AES256, Blowfish, CAST, DES, 3DES

.PARAMETER HashAlgorithm
The hash algorithm which should be used for the VPN tunnel.
Possible values are: None, MD5, SHA1

.EXAMPLE
.\AzureConnectivityAgent.ps1 johndoe.pfx -PfxFilePassword mysecret -
ProfileDescription ConnectivityAgent -ServerAddress vpn.acme.org -TunnelMode
TCP -EncryptionAlgorithm AES256 -HashAlgorithm SHA1

Generates a new configuration file based on a given (password-protected) PFX
container file and a given set of VPN profile parameters.
After the configuration file has been generated, the installation process
will be started immediately.

.EXAMPLE
.\AzureConnectivityAgent.ps1 -ConfigureOnly johndoe.pfx -PfxFilePassword
mysecret

Generates a new configuration file based on a given (non-password protected)
PFX container file.
VPN profile parameters are collected interactively at the command prompt by
answering the respective questions.

.EXAMPLE
.\AzureConnectivityAgent.ps1 -InstallOnly johndoe.pfx -PfxFilePassword
mysecret

Starts the installation process by using a given (password-protected) PFX
container file and the current configuration file.
#>

param(
[switch]$ConfigureOnly,
[switch]$InstallOnly,
[Parameter(Mandatory=$True)]
[string]$PfxFileName,
[string]$PfxFilePassword,
[string]$ProfileDescription,
```

```
[string]$ServerAddress,
[ValidateSet('TCP','UDP','Hybrid')]
[string]$TunnelMode,
[ValidateSet('None','AES','AES256','Blowfish','CAST','DES','3DES')]
[string]$EncryptionAlgorithm,
[ValidateSet('None','MD5','SHA1')]
[string]$HashAlgorithm
)

$placeholders       = "&:description:&", "&:server:&",
"&:mode:&","&:encryption:&", "&:hash:&",
"&:certname:&","&:certserialnumber:&", "&:certissuer:&", "&:certissuerX500:&"
$phDescription      = $placeholders[0]
$phServer           = $placeholders[1]
$phMode             = $placeholders[2]
$phEncryption       = $placeholders[3]
$phHash             = $placeholders[4]
$phCertname         = $placeholders[5]
$phCertserialnumber = $placeholders[6]
$phCertissuer       = $placeholders[7]
$phCertissuerX500   = $placeholders[8]

$scriptName         = $MyInvocation.MyCommand.Name
$scriptDir          = Split-Path -Parent -Path
$MyInvocation.MyCommand.Definition
$pfxFullFilePath    = Join-Path $scriptDir $PfxFileName
$infFullFilePath    = Join-Path $scriptDir "customer.inf"

$tunnelModeId          = 0
$encryptionAlgorithmId = 0
$hashAlgorithmId       = 0
$certName              = ""
$certSerial            = ""
$certIssuer            = ""
$certIssuerName        = ""

function AskTunnelMode()
{
    $result = 3
    $c1 = New-Object System.Management.Automation.Host.ChoiceDescription
"Reliability (&TCP)","Reliability (TCP)"
    $c2 = New-Object System.Management.Automation.Host.ChoiceDescription
"Response (&UDP)","Response (UDP)"
    $c3 = New-Object System.Management.Automation.Host.ChoiceDescription
"Optimized (&Hybrid)","Optimized (Hybrid)"
    $choices =
```

```
[System.Management.Automation.Host.ChoiceDescription[]]($c1,$c2,$c3)
    $caption = "Tunnel mode"
    $result = $Host.UI.PromptForChoice($caption, "", $choices, 2) # 'Hybrid'
is default
    If($result -eq 0) { $result = 1 } # TCP
    ElseIf($result -eq 1) { $result = 2 } # UDP
    ElseIf($result -eq 2) { $result = 3 } # Hybrid
    return $result
}

function ParseTunnelMode($tm)
{
    $str = $tm.ToUpperInvariant()
    If($str -eq "TCP")
    {
        return 1
    }
    ElseIf($str -eq "UDP")
    {
        return 2
    }
    ElseIf($str -eq "HYBRID")
    {
        return 3
    }
    Else
    {
        Throw "Invalid Tunnel mode"
    }
}

function AskEncryptionAlgorithm()
{
    $result = 4
    $c1 = New-Object System.Management.Automation.Host.ChoiceDescription
"&None","No encryption"
    $c2 = New-Object System.Management.Automation.Host.ChoiceDescription
"&AES","AES"
    $c3 = New-Object System.Management.Automation.Host.ChoiceDescription
"AES&256","AES256"
    $c4 = New-Object System.Management.Automation.Host.ChoiceDescription
"&Blowfish","Blowfish"
    $c5 = New-Object System.Management.Automation.Host.ChoiceDescription
"&CAST","CAST"
    $c6 = New-Object System.Management.Automation.Host.ChoiceDescription
"&DES","DES"
```

```
    $c7 = New-Object System.Management.Automation.Host.ChoiceDescription
"&3DES","3DES"
    $choices =
[System.Management.Automation.Host.ChoiceDescription[]]($c1,$c2,$c3,$c4,$c5,$
c6,$c7)
    $caption = "Encryption algorithm"
    $result = $Host.UI.PromptForChoice($caption, "", $choices, 1) # 'AES' is
default
    If($result -eq 0) { $result = 1  } # None
    ElseIf($result -eq 1) { $result = 4  } # AES
    ElseIf($result -eq 2) { $result = 64 } # AES256
    ElseIf($result -eq 3) { $result = 16 } # Blowfish
    ElseIf($result -eq 4) { $result = 8  } # CAST
    ElseIf($result -eq 5) { $result = 32 } # DES
    ElseIf($result -eq 6) { $result = 2  } # 3DES
    return $result
}

function ParseEncryptionAlgorithm($ea)
{
    $str = $ea.ToUpperInvariant()
    If($str -eq "NONE")
    {
        return 1
    }
    ElseIf($str -eq "AES")
    {
        return 4
    }
    ElseIf($str -eq "AES256")
    {
        return 64
    }
    ElseIf($str -eq "BLOWFISH")
    {
        return 16
    }
    ElseIf($str -eq "CAST")
    {
        return 8
    }
    ElseIf($str -eq "DES")
    {
        return 32
    }
    ElseIf($str -eq "3DES")
```

```
    {
        return 2
    }
    Else
    {
        Throw "Invalid Encryption algorithm"
    }
}

function AskHashAlgorithm()
{
    $result = 1
    $c1 = New-Object System.Management.Automation.Host.ChoiceDescription
"&None","No hash"
    $c2 = New-Object System.Management.Automation.Host.ChoiceDescription
"&MD5","MD5"
    $c3 = New-Object System.Management.Automation.Host.ChoiceDescription
"&SHA1","SHA1"
    $choices =
[System.Management.Automation.Host.ChoiceDescription[]]($c1,$c2,$c3)
    $caption = "Hash algorithm"
    $result = $Host.UI.PromptForChoice($caption, "", $choices, 1) # 'MD5' is
default
    If($result -eq 0) { $result = 3 } # None
    ElseIf($result -eq 1) { $result = 1 } # MD5
    ElseIf($result -eq 2) { $result = 2 } # SHA1
    return $result
}

function ParseHashAlgorithm($ha)
{
    $str = $ha.ToUpperInvariant()
    If($str -eq "NONE")
    {
        return 3
    }
    ElseIf($str -eq "MD5")
    {
        return 1
    }
    ElseIf($str -eq "SHA1")
    {
        return 2
    }
    Else
    {
```

```
        Throw "Invalid Hash algorithm"
    }
}


function ExtractCertificateData()
{
    If($PfxFilePassword)
    {
        $cert = New-Object
System.Security.Cryptography.X509Certificates.X509Certificate2($pfxFullFilePa
th, $PfxFilePassword)
    }
    Else
    {
        $cert = New-Object
System.Security.Cryptography.X509Certificates.X509Certificate2($pfxFullFilePa
th)
    }

    Write-Host "========================"
    Write-Host "Certificate information:"
    Write-Host "========================"

    $subject = $cert.Subject
    Write-Host -NoNewLine "Subject:       "
    Write-Host -fore Yellow $subject

    $script:certName =
$cert.GetNameInfo([System.Security.Cryptography.X509Certificates.X509NameType
]::SimpleName, $False)
    Write-Host -NoNewline "Name:          "
    Write-Host -fore Yellow $certName

    $script:certIssuer =
$cert.GetNameInfo([System.Security.Cryptography.X509Certificates.X509NameType
]::SimpleName, $True)
    Write-Host -NoNewline "Issuer         "
    Write-Host -fore Yellow $certIssuer

    $script:certIssuerName = $cert.GetIssuerName()
    Write-Host -NoNewline "Issuer name:   "
    Write-Host -fore Yellow $certIssuerName

    $notbefore = $cert.NotBefore
    Write-Host -NoNewLine "Valid from:    "
```

```
    Write-Host -fore Yellow $notbefore

    $notafter = $cert.NotAfter
    Write-Host -NoNewLine "Valid to:      "
    Write-Host -fore Yellow $notafter

    $script:certSerial = $cert.GetSerialNumberString().ToLowerInvariant()
    Write-Host -NoNewline "Serial number: "
    Write-Host -fore Yellow $certSerial

    $thumbprint = $cert.Thumbprint
    Write-Host -NoNewLine "Thumbprint:    "
    Write-Host -fore Yellow $thumbprint
    Write-Host
}

function GenerateConfigFile()
{
    $template =
"OyAtLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0t
LS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLQ0KOyBjdXN0b21lci5pbmYNCjsNC
jsgQ3VzdG9tZXIgSW5zdGFsbCBGaWxlDQo7DQo7IENvcHlyaWdodCAoQykgMjAwOC0yMDE0IEJhcn
JhY3VkYSBOZXR3b3JrcywgSW5jLg0KOw0KOyBGb3IgZGV0YWlsZWQgaW5mb3JtYXRpb24gcGxlYXN
lIGNvbnNpZGVyIHRoZSBCYXJyYWN1ZGEgVGVjaExpbjhcnkNCjsgLS0tLS0tLS0tLS0tLS0tLS0t
LS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tL
S0tLS0tLS0tLS0tLS0NCg0KW3ZlcnNpb25dDQpzaWduYXR1cmUgPSAiJFdpbmRvd3MgTlQkIg
0KcHJvdmlkZXIgID0gJXBoJQ0KDQpbTWFudWZhY3R1cmVyXQ0KJVBoaW9uJSA9IEJhcnJhY3VkYQ0
KDQpbRGVmYXVsdEluc3RhbGxdDQpDb3B5RmlsZXMgPSBQaGlvbkN1c3RvbWVyQ29weUZpbGVzDQpB
ZGRSZWcgICAgPSBQaGlvbkN1c3RvbWVyUmVnDQoNCltEZWZhdWx0VW5pbnN0YWxsXQ0KRGVsRmlsZ
XMgPSBQaGlvbkN1c3RvbWVyQ29weUZpbGVzDQpEZWxSZWcgICA9IFBoaW9uQ3VzdG9tZXJSZWcNCg
0KOyAtLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0
tLS0tLS0tLS0tLS0tLS0tLS0tLQ0KOyAxLCBDdXN0b21lciBBcmVhDQo7IC0tLS0tLS0tLS0tLS0t
LS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tL
S0tLS0tLS0tDQpbUGhpb25DdXN0b21lckNvcHlGaWxlc10NCjsgZGVzdGluYXRpb24tZmlsZS1uYW
1lWyxzb3VyY2UtZmlsZS1uYW1lXVssdGVtcG9yYXJ5LWZpbGUtbmFtZV1bLGZsYWddDQpjdXN0b21
lci5pbmYsLCwyICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICA7IGlt
cG9ydGFudCwgZG8gbm90IHJlbW92ZQ0KDQoNCjsgLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tL
S0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0NCjsgMiwgQ3
VzdG9tZXIgQXJlYQ0KOyAtLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0
tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLQ0KW1BoaW9uQ3VzdG9tZXJSZWddDQo7
IHJlZy1yb290LCBbc3Via2V5XSwgW3ZhbHVlLWVudHJ5LW5hbWVdLCBbZmxhZ3NdLCBbdmFsdWVdD
QpIS1UsIC5ERUZBVUxUXFNvZnR3YXJlXFBoaW9uXHBoaW9udnBuLCBDdXN0b21lcklORiwgMHgwMD
AwMDAwMCwgIiU2NTYwMCVcY3VzdG9tZXIuaW5mIiA7IGltcG9ydGFudCwgZG8gbm90IHJlbW92ZQ0
KDQo7IFByb2ZpbGUgMSB3aXRoIE1pY3Jvc29mdCBDZXJ0aWZpY2F0ZSBTdG9yZS1saW5rZWQgWC41
MDkgY2VydGlmaWNhdGUNCkhLVSwgLkRFRkFVTFRcU29mdHdhcmVcUGhpb25ccGhpb252cG5cUHJvZ
mlsZVwxLCBBdXRoVHlwZSwgICAgICAgICAgICAweDAwMDEwMDAxLCAxDQpIS1UsIC5ERUZBVUxUXF
```

```
NvZnR3YXJlXFBoaW9uXHBoaW9udnBuXFByb2ZpbGVcMSwgQXV0aFVzZXIsICAgICAgICAgICAgMHg
wMDAxMDAwMSwgMA0KSEtVLCAuREVGQVVMVFxTb2Z0d2FyZVxQaGlvblxwaGlvbnZwblxQcm9maWxl
XDEsIGF1dG9UcnVzdFg1MDlTZXJ2ZXIsIDB4MDAwMTAwMDEsIDENCkhLVSwgLkRFRkFVTFRcU29md
HdhcmVcUGhpb25ccGhpb252cG5cUHJvZmlsZVwxLCBDcnlwdG9Nb2RlLCAgICAgICAgICAweDAwMD
EwMDAxLCAxDQpIS1UsIC5ERUZBVUxUXFNvZnR3YXJlXFBoaW9uXHBoaW9udnBuXFByb2ZpbGVcMSw
gRGVmYXVsdCwgICAgICAgICAgICAgMHgwMDAxMDAwMSwgMQ0KSEtVLCAuREVGQVVMVFxTb2Z0d2Fy
ZVxQaGlvblxwaGlvbnZwblxQcm9maWxlXDEsIGRoY3AsICAgICAgICAgICAgICAgIDB4MDAwMTAwM
DEsIDENCkhLVSwgLkRFRkFVTFRcU29mdHdhcmVcUGhpb25ccGhpb252cG5cUHJvZmlsZVwxLCBLZX
lTcGVjLCAgICAgICAgICAgICAweDAwMDEwMDAxLCAxDQpIS1UsIC5ERUZBVUxUXFNvZnR3YXJlXFB
oaW9uXHBoaW9udnBuXFByb2ZpbGVcMSwgUHJpdmF0ZUVuY3J5cHQsICAgICAgMHgwMDAxMDAwMSwg
MQ0KSEtVLCAuREVGQVVMVFxTb2Z0d2FyZVxQaGlvblxwaGlvbnZwblxQcm9maWxlXDEsIFNob3dQb
3B1cCwgICAgICAgICAgIDB4MDAwMTAwMDEsIDANCkhLVSwgLkRFRkFVTFRcU29mdHdhcmVcUGhpb2
5ccGhpb252cG5cUHJvZmlsZVwxLCBzdG9yZSwgICAgICAgICAgICAgICAweDAwMDAwMDAwLCAiTVk
iDQpIS1UsIC5ERUZBVUxUXFNvZnR3YXJlXFBoaW9uXHBoaW9udnBuXFByb2ZpbGVcMSwgU3RvcmVG
bGFncywgICAgICAgICAgMHgwMDAxMDAwMSwgMHgwMDAyMDAwMA0KSEtVLCAuREVGQVVMVFxTb2Z0d
2FyZVxQaGlvblxwaGlvbnZwblxQcm9maWxlXDEsIGRlc2NyaXB0aW9uLCAgICAgICAgIDB4MDAwMD
AwMDAsICImOmRlc2NyaXB0aW9uOiYiICAgICAgIDsgUHJvZmlsZSBkZXNjcmlwdGlvbiAoZS5nLiB
kZWZhdWx0KQ0KSEtVLCAuREVGQVVMVFxTb2Z0d2FyZVxQaGlvblxwaGlvbnZwblxQcm9maWxlXDEs
IHNlcnZlciwgICAgICAgICAgICAgIDB4MDAwMDAwMDAsICImOnNlcnZlcjomIiAgICAgICAgICAgI
DsgVlBOIFNlcnZlciBBZGRyZXNzIChlLmcuIDEwLjE2LjczLjE2MykNCkhLVSwgLkRFRkFVTFRcU2
9mdHdhcmVcUGhpb25ccGhpb252cG5cUHJvZmlsZVwxLCBtb2RlLCAgICAgICAgICAgICAgICAweDA
wMDEwMDAxLCAmOm1vZGU6JiAgICAgICAgICAgICAgICA7IDE9UmVsaWFiaWxpdHkgKFRDUCksIDI9
UmVzcG9uc2UgKFVEUCksIDM9T3B0aW1pemVkIChIeWJyaWQpDQpIS1UsIC5ERUZBVUxUXFNvZnR3Y
XJlXFBoaW9uXHBoaW9udnBuXFByb2ZpbGVcMSwgZW5jcnlwdGlvbiwgICAgICAgICAgMHgwMDAxMD
AwMSwgJjplbmNyeXB0aW9uOiYgICAgICAgICAgOyAxPU5vbmUsIDI9M0RFUywgND1BRVMsIDg9Q2F
zdCwgMTY9Qmxvd2Zpc2gsIDMyPURFUywgNjQ9QUVTMjU2DQpIS1UsIC5ERUZBVUxUXFNvZnR3YXJl
XFBoaW9uXHBoaW9udnBuXFByb2ZpbGVcMSwgaGFzaCwgICAgICAgICAgICAgICAgMHgwMDAxMDAwM
SwgJjpoYXNoOiYgICAgICAgICAgICAgICAgOyAxPU1ENSwgMj1TSEExLCAzPU5vbmUNCkhLVSwgLk
RFRkFVTFRcU29mdHdhcmVcUGhpb25ccGhpb252cG5cUHJvZmlsZVwxLCBjZXJ0bmFtZSwgICAgICA
gICAgICAweDAwMDAwMDAwLCAiJjpjZXJ0bmFtZTomIiAgICAgICAgICA7IENlcnRpZmljYXRlIE5h
bWUgKGUuZy4gam9obmRvZSkNCkhLVSwgLkRFRkFVTFRcU29mdHdhcmVcUGhpb25ccGhpb252cG5cU
HJvZmlsZVwxLCBjZXJ0c2VyaWFsbnVtYmVyLCAgICAweDAwMDAwMDAwLCAiJjpjZXJ0c2VyaWFsbn
VtYmVyOiYiICA7IENlcnRpZmljYXRlIFNlcmlhbCBOdW1iZXIgKGUuZy4gMTAyMmQ5NzIwMDAwMDA
wMDNhYjQpDQpIS1UsIC5ERUZBVUxUXFNvZnR3YXJlXFBoaW9uXHBoaW9udnBuXFByb2ZpbGVcMSwg
Y2VydGlzc3VlciwgICAgICAgICAgMHgwMDAwMDAwMCwgIiY6Y2VydGlzc3VlcjomIiAgICAgICAgO
yBDZXJ0aWZpY2F0ZSBJc3N1ZXIgKGUuZy4gY2VydHNydikNCkhLVSwgLkRFRkFVTFRcU29mdHdhcm
VcUGhpb25ccGhpb252cG5cUHJvZmlsZVwxLCBjZXJ0aXNzdWVyWDUwMCwgICAgICAweDAwMDAwMDA
wLCAiJjpjZXJ0aXNzdWVyWDUwMDomIiAgICA7IENlcnRpZmljYXRlIElzc3VlciBOYW1lIChlLmcu
IERDPWxvY2FsLCBEQz1hY21lLCBDTj1jZXJ0c3J2KQ0KDQoNCjsgZ2xvYmFsIHNldHRpbmdzDQpIS
1UsIC5ERUZBVUxUXFNvZnR3YXJlXFBoaW9uXHBoaW9udnBuXHNldHRpbmdzLCBkaXJlY3RhY2Nlc3
MsIDB4MDAwMTAwMDEsIDENCg0KDQo7IC0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0
tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tDQo7IDMsIEN1c3RvbWVy
IEFyZWENCjsgLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tL
S0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0NCltTb3VyY2VEaXNrc0ZpbGVzXQ0KOyBGaWxlcy
Bmb3IgZGlzayBCYXJyYWN1ZGEgQ3VzdG9tZXIgRmlsZXMgIzENCjsgZmlsZW5hbWUgPSBkaXNraWR
bLFsgc3ViZGlyXVssIHNpemVdXSINCmN1c3RvbWVyLmluZiwsLDENCg0KDQo7LS0tLS0tLS0tLS0t
```

```
LS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tL
S0tLS0tLQ0KOyBEbyBub3QgY2hhbmdlIGFueSBhdHRyaWJ1dGUgYmV5b25kIHRoaXMgbGluZSENCj
sNCltEZXN0aW5hdGlvbkRpcnNdDQpQaGlvbkN1c3RvbWVyQ29weUZpbGVzID0gNjU2MDANCg0KW1N
vdXJjZURpc2tzTmFtZXNdDQoxID0gJURpc2tJZDElLCwsIiINCg0KOy0tLS0tLS0tLS0tLS0tLS0t
LS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tL
S0NCjsgTG9jYWxpemFibGUgU3RyaW5ncw0KOw0KW1N0cmluZ3NdDQpwaCAgICAgICAgICAgICAgIC
AgPSAiQmFycmFjdWRhIg0KRGlzcGxheUNsYXNzTmFtZSAgID0gIkJhcnJhY3VkYSBDdXN0b21lciB
GaWxlcyINClBoaW9uICAgICAgICAgICAgICA9ICJCYXJyYWN1ZGEgTmV0d29ya3MsIEluYy4iDQoq
UGhpb25kLkRldmljZURlc2MgPSAiQmFycmFjdWRhIEN1c3RvbWVyIEZpbGVzIg0KUGhpb24uRGV2a
WNlRGVzYyAgID0gIkJhcnJhY3VkYSBDdXN0b21lciBGaWxlcyINCipQaGlvbi5EZXZpY2VEZXNjIC
A9ICJCYXJyYWN1ZGEgQ3VzdG9tZXIgRmlsZXMiDQpwaGlvbnZwbi5TZXJ2aWNlLkRpc3BOYW1lID0
gIkJhcnJhY3VkYSBDdXN0b21lciBGaWxlcyINCkRpc2tJZDEgICAgICAgICAgICAgICAgICAgPSAi
QmFycmFjdWRhIEN1c3RvbWVyIEZpbGVzIERpc2sgIzEiDQo="
    $data = [System.Convert]::FromBase64String($template)
    Set-Content -Path $inffullFilePath -Value $data -Encoding Byte
    (Get-Content $inffullFilePath) | Foreach-Object {$_ -replace
$phDescription, $ProfileDescription} | Set-Content $inffullFilePath
    (Get-Content $inffullFilePath) | Foreach-Object {$_ -replace $phServer,
$ServerAddress} | Set-Content $inffullFilePath
    (Get-Content $inffullFilePath) | Foreach-Object {$_ -replace $phMode,
$tunnelModeId} | Set-Content $inffullFilePath
    (Get-Content $inffullFilePath) | Foreach-Object {$_ -replace
$phEncryption, $encryptionAlgorithmId} | Set-Content $inffullFilePath
    (Get-Content $inffullFilePath) | Foreach-Object {$_ -replace $phHash,
$hashAlgorithmId} | Set-Content $inffullFilePath
    (Get-Content $inffullFilePath) | Foreach-Object {$_ -replace $phCertname,
$certName} | Set-Content $inffullFilePath
    (Get-Content $inffullFilePath) | Foreach-Object {$_ -replace
$phCertserialnumber, $certSerial} | Set-Content $inffullFilePath
    (Get-Content $inffullFilePath) | Foreach-Object {$_ -replace
$phCertissuer, $certIssuer} | Set-Content $inffullFilePath
    (Get-Content $inffullFilePath) | Foreach-Object {$_ -replace
$phCertissuerX500, $certIssuerName} | Set-Content $inffullFilePath
}

function GetProfileData()
{
    if(!$ProfileDescription -or !$ServerAddress -or !$TunnelMode -or
!$EncryptionAlgorithm -or !$HashAlgorithm)
    {
        Write-Host "Please answer the following questions to complete your
configuration file:"

        If(!$ProfileDescription)
        {
            Write-Host -NoNewLine "Profile description: "
```

```
            $script:ProfileDescription = Read-Host
            If(!$ProfileDescription)
            {
                 $script:ProfileDescription = "default"
            }
        }

        If(!$ServerAddress)
        {
            Write-Host -NoNewLine "VPN server address/hostname: "
            $script:ServerAddress = Read-Host
            If(!$ServerAddress)
            {
                $script:ServerAddress = "vpnserver"
            }
        }

        If(!$TunnelMode)
        {
            $script:tunnelModeId = AskTunnelMode
        }

        If(!$EncryptionAlgorithm)
        {
            $script:encryptionAlgorithmId = AskEncryptionAlgorithm
        }

        If(!$HashAlgorithm)
        {
            $script:hashAlgorithmId = AskHashAlgorithm
        }
    }
}

function IsConfigFileValid()
{
    If(!(Test-Path $infFullFilePath))
    {
        return $False
    }

    $file = Get-Content $infFullFilePath
    Foreach($element in $placeholders)
    {
        $contains = $file | %{$_ -match $element}
        if($contains -contains $True)
```

```
        {
            return $False
        }
    }

    return $True
}


function ImportCertificate()
{
    If(Get-Command -CommandType Cmdlet -Module PKI Import-PfxCertificate -
errorAction SilentlyContinue)
    {
        # starting with Windows Server 2012, private keys may also be
protected to security principals
        Write-Host "Using 'Import-PfxCertificate' for certificate import..."
        If($PfxFilePassword)
        {
            $pfxSecureFilePassword = ConvertTo-SecureString $PfxFilePassword
-AsPlainText -Force
            Import-PfxCertificate $PfxFileName -CertStoreLocation
cert:\localMachine\my -Password $pfxSecureFilePassword
        }
        Else
        {
            Import-PfxCertificate $PfxFileName -CertStoreLocation
cert:\localMachine\my
        }
    }
    Else
    {
        Write-Host "Using 'certutil' for certifcate import..."
        If($PfxFilePassword)
        {
            certutil -f -p $PfxFilePassword -importPFX $PfxFileName
        }
        Else
        {
            certutil -f -importPFX $PfxFileName
        }

        If($LASTEXITCODE -ne 0)
        {
            Throw ""
        }
```

```
    }
}

function StartSetup()
{
    .\setup.exe /s /v"/qr CUSTOMER_INF=customer.inf PROGTYPE=VPN"
    # .\setup.exe /s /v"/qr CUSTOMER_INF=customer.inf PROGTYPE=VPN
/Lecumwvario setup.log"
}

function RemoveFileIfExists($file)
{
    Remove-Item -Force $file -errorAction SilentlyContinue
}

function RunConfigMode()
{
    Write-Host "Entering configuration mode..."

    Try
    {
        ExtractCertificateData
        Try
        {
            GetProfileData
            Try
            {
                GenerateConfigFile
                Write-Host "Configuration file has successfully been
generated."
            }
            Catch
            {
                RemoveFileIfExists($infFullFilePath)
                Write-Error "Configuration file could not be generated!"
            }
        }
        Catch
        {
            Write-Error "Profile data could not be retrieved!"
        }
    }
    Catch
    {
        Write-Error "Certificate data could not be extracted!"
    }
```

```
}

function RunInstallMode()
{
    Write-Host "Entering install mode..."

    If(!(IsConfigFileValid))
    {
        Write-Error "Invalid configuration file!"
        return
    }

    Try
    {
        ImportCertificate
        Write-Host "Certificate has successfully been imported."

        Try
        {
            StartSetup
        }
        Catch
        {
            Write-Error "Setup could not be started!"
        }
    }
    Catch
    {
        Write-Error "Certificate could not be imported!"
    }
}

Write-Host
"================================================================================
==="
Write-Host "Welcome to the Barracuda Azure Connectivity Agent Configuration
and Setup Tool!"
Write-Host
"================================================================================
==="
Write-Host

If($TunnelMode)
{
    $tunnelModeId = ParseTunnelMode($TunnelMode)
}
```

```
If($EncryptionAlgorithm)
{
    $encryptionAlgorithmId = ParseEncryptionAlgorithm($EncryptionAlgorithm)
}
If($HashAlgorithm)
{
    $hashAlgorithmId = ParseHashAlgorithm($HashAlgorithm)
}

If($ConfigureOnly -and !$InstallOnly)
{
    RunConfigMode
}
ElseIf(!$ConfigureOnly -and $InstallOnly)
{
    RunInstallMode
}
Else
{
    RunConfigMode
    RunInstallMode
}

# SIG # Begin signature block
# MIIY+gYJKoZIhvcNAQcCoIIY6zCCGOcCAQExCzAJBgUrDgMCGgUAMGkGCisGAQQB
# gjcCAQSgWzBZMDQGCisGAQQBgjcCAR4wJgIDAQAABBAfzDtgWUsITrck0sYpfvNR
# AgEAAgEAAgEAAgEAAgEAMCEwCQYFKw4DAhoFAAQUb+CXvjGrJJqKUJB+iqTELi58
# 1dWgghPqMIID7jCCA1egAwIBAgIQfpPr+3zGTlnqS5p31Ab8OzANBgkqhkiG9w0B
# AQUFADCBizELMAkGA1UEBhMCWkExFTATBgNVBAgTDFdlc3Rlcm4gQ2FwZTEUMBIG
# A1UEBxMLRHVyYmFudmlsbGUxDzANBgNVBAoTBlRoYXd0ZTEdMBsGA1UECxMUVGhh
# d3RlIENlcnRpZmljYXRpb24xHzAdBgNVBAMTFlRoYXd0ZSBUaW1lc3RhbXBpbmcg
# Q0EwHhcNMTIxMjIxMDAwMDAwWhcNMjAxMjMwMjM1OTU5WjBeMQswCQYDVQQGEwJV
# UzEdMBsGA1UEChMUU3ltYW50ZWMgQ29ycG9yYXRpb24xMDAuBgNVBAMTJ1N5bWFu
# dGVjIFRpbWUgU3RhbXBpbmcgU2VydmljZXMgQ0EgLSBHMjCCASIwDQYJKoZIhvcN
# AQEBBQADggEPADCCAQoCggEBALGss0lUS5ccEgrYJXmRIlcqb9y4JsRDc2vCvy5Q
# WvsUwnaOQwElQ7Sh4kX06Ld7w3TMIte0lAAC903tv7S3RCRrzV9FO9FEzkMScxeC
# i2m0K8uZHqxyGyZNcR+xMd37UWECU6aq9UksBXhFpS+JzueZ5/6M4lc/PcaS3Er4
# ezPkeQr78HWIQZz/xQNRmarXbJ+TaYdlKYOFwmAUxMjJOxTawIHwHw103pIiq8r3
# +3R8J+b3Sht/p8OeLa6K6qbmqicWfWH3mHERvOJQoUvlXfrlDqcsn6plINPYlujI
# fKVOSET/GeJEB5IL12iEgF1qeGRFzWBGflTBE3zFefHJwXECAwEAAaOB+jCB9zAd
# BgNVHQ4EFgQUX5r1blzMzHSa1N197z/b7EyALt0wMgYIKwYBBQUHAQEEJjAkMCIG
# CCsGAQUFBzABhhZodHRwOi8vb2NzcC50aGF3dGUuY29tMBIGA1UdEwEB/wQIMAYB
# Af8CAQAwPwYDVR0fBDgwNjA0oDKgMIYuaHR0cDovL2NybC50aGF3dGUuY29tL1Ro
# YXd0ZVRpbWVzdGFtcGluZ0NBLmNybDATBgNVHSUEDDAKBggrBgEFBQcDCDAOBgNV
# HQ8BAf8EBAMCAQYwKAYDVR0RBCEwH6QdMBsxGTAXBgNVBAMTEFRpbWVTdGFtcC0y
# MDQ4LTEwDQYJKoZIhvcNAQEFBQADgYEAAwmbj3nvf1kwqu9otfrjCR27T4IGXTdf
```

```
# plKfFo3qHJIJRG71betYfDDo+WmNI3MLEm9Hqa45EfgqsZuwGsOO61mWAK3ODE2y
# 0DGmCFwqevzieh1XTKhlGOl5QGIllm7HxzdqgyEIjkHq3dlXPx13SYcqFgZepjhq
# IhKjURmDfrYwggSjMIIDi6ADAgECAhAOz/Q4yP6/NW4E2GqYGxpQMA0GCSqGSIb3
# DQEBBQUAMF4xCzAJBgNVBAYTAlVTMR0wGwYDVQQKExRTeW1hbnRlYyBDb3Jwb3Jh
# dGlvbjEwMC4GA1UEAxMnU3ltYW50ZWMgVGltZSBTdGFtcGluZyBTZXJ2aWNlcyBD
# QSAtIEcyMB4XDTEyMTAxODAwMDAwMFoXDTIwMTIyOTIzNTk1OVowYjELMAkGA1UE
# BhMCVVMxHTAbBgNVBAoTFFN5bWFudGVjIENvcnBvcmF0aW9uMTQwMgYDVQQDEytT
# eW1hbnRlYyBUaW1lIFN0YW1waW5nIFNlcnZpY2VzIFNpZ25lciAtIEc0MIIBIjAN
# BgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAomMLOUS4uyOnREm7Dv+h8GEKU5Ow
# mNutLA9KxW7/hjxTVQ8VzgQ/K/2plpbZvmF5C1vJTIZ25eBDSyKV7sIrQ8Gf2Gi0
# jkBP7oU4uRHFI/JkWPAVMm9OV6GuiKQC1yoezUvh3WPVF4kyW7BemVqonShQDhfu
# ltthO0VRHc8SVguSR/yrrvZmPUescHLnkudfzRC5xINklBm9JYDh6NIipdC6Anqh
# d5NbZcPuF3S8QYYq3AhMjJKMkS2ed0QfaNaodHfbDlsyi1aLM73ZY8hJnTrFxeoz
# C9Lxoxv0i77Zs1eLO94Ep3oisiSuLsdwxb5OgyYI+wu9qU+ZCOEQKHKqzQIDAQAB
# o4IBVzCCAVMwDAYDVR0TAQH/BAIwADAWBgNVHSUBAf8EDDAKBggrBgEFBQcDCDAO
# BgNVHQ8BAf8EBAMCB4AwcwYIKwYBBQUHAQEEZzBlMCoGCCsGAQUFBzABhh5odHRw
# Oi8vdHMtb2NzcC53cy5zeW1hbnRlYy5jb20wNwYIKwYBBQUHMAKGK2h0dHA6Ly90
# cy1haWEud3Muc3ltYW50ZWMuY29tL3Rzcy1jYS1nMi5jZXIwPAYDVR0fBDUwMzAx
# oC+gLYYraHR0cDovL3RzLWNybC53cy5zeW1hbnRlYy5jb20vdHNzLWNhLWcyLmNy
# bDAoBgNVHREEITAfpB0wGzEZMBcGA1UEAxMQVGltZVN0YW1wLTIwNDgtMjAdBgNV
# HQ4EFgQURsZpow5KFB7VTNpSYxc/Xja8DeYwHwYDVR0jBBgwFoAUX5r1blzMzHSa
# 1N197z/b7EyALt0wDQYJKoZIhvcNAQEFBQADggEBAHg7tJEqAEzwj2IwN3ijhCcH
# bxiy3iXcoNSUA6qGTiWfmkADHN3O43nLIWgG2rYytG2/9CwmYzPkSWRtDebDZw73
# BaQ1bHyJFsbpst+y6d0gxnEPzZV03LZc3r03H0N45ni1zSgEIKOq8UvEiCmRDoDR
# EfzdXHZuT14ORUZBbg2w6jiasTraCXEQ/Bx5tIB7rGn0/Zy2DBYr8X9bCT2bW+IW
# yhOBbQAuOA2oKY8s4bL0WqkBrxWcLC9JG9siu8P+eJRRw4axgohd8D20UaF5Mysu
# e7ncIAkTcetqGVvP6KUwVyyJST+5z3/Jvz4iaGNTmr1pdKzFHTx/kuDDvBzYBHUw
# ggU/MIIEJ6ADAgECAhBE7dMvNLbg1XINy7LQmAfFMA0GCSqGSIb3DQEBCwUAMIG0
# MQswCQYDVQQGEwJVUzEXMBUGA1UEChMOVmVyaVNpZ24sIEluYy4xHzAdBgNVBAsT
# FlZlcmlTaWduIFRydXN0IE5ldHdvcmsxOzA5BgNVBAsTMlRlcm1zIG9mIHVzZSBh
# dCBodHRwczovL3d3dy52ZXJpc2lnbi5jb20vcnBhIChjKTEwMS4wLAYDVQQDEyVW
# ZXJpU2lnbiBDbGFzcyAzIENvZGUgU2lnbmluZyAyMDEwIENBMB4XDTE0MTExMzAw
# MDAwMFoXDTE3MDIxMTIzNTk1OVowcTELMAkGA1UEBhMCQVQxDjAMBgNVBAgTBVR5
# cm9sMRIwEAYDVQQHEwlJbm5zYnJ1Y2sxHjAcBgNVBAoUFUJhcnJhY3VkYSBOZXR3
# b3JrcyBBRzEeMBwGA1UEAxQVQmFycmFjdWRhIE5ldHdvcmtzIEFHMIIBIjANBgkq
# hkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA6Va3PcbI+4j/Kx2xFX8Uqm/hjYTJtCWZ
# 3JoNhgldIQujCsevSCuoXHsjiVlHIK6ljWF1im2BdoEeNXkQYG7WA1lU/BGNpVVA
# gfvaxaTj8L1bjpa7U4eiqmoSSQ4/u29cHB0CNStyeMviewN/varSHZgbB39hW3yX
# zfGIYGZvFRoUcn4XsuRVxsVMonzCA96G8G8cIpJ8a7z6lNwrTvhihxS5mNqspXZq
# Ctm9Fzmv0W8p2XpeZpDGWdzsnh62DAKUe0TBByW3rfdXDi7jh9jR55tOxFvbBC+s
# jxIw2b/CPtnDP+LSN9FI6R+lmZsfNbo9SDTOoMGNZgpCVcRv4ACUXwIDAQABo4IB
# jTCCAYkwCQYDVR0TBAIwADAOBgNVHQ8BAf8EBAMCB4AwKwYDVR0fBCQwIjAgoB6g
# HIYaaHR0cDovL3NmLnN5bWNiLmNvbS9zZi5jcmwwZgYDVR0gBF8wXTBbBgtghkgB
# hvhFAQcXAzBMMCMGCCsGAQUFBwIBFhdodHRwczovL2Quc3ltY2IuY29tL2NwczAl
# BggrBgEFBQcCAjAZDBdodHRwczovL2Quc3ltY2IuY29tL3JwYTATBgNVHSUEDDAK
# BggrBgEFBQcDAzBXBggrBgEFBQcBAQRLMEkwHwYIKwYBBQUHMAGGE2h0dHA6Ly9z
```

```
# Zi5zeW1jZC5jb20wJgYIKwYBBQUHMAKGGmh0dHA6Ly9zZi5zeW1jYi5jb20vc2Yu
# Y3J0MB8GA1UdIwQYMBaAFM+Zqep7JvRLyY6P1/AFJu/j0qedMB0GA1UdDgQWBBSS
# 3UFsXosnT2I3GWzEANOTjzWcszARBglghkgBhvhCAQEEBAMCBBAwFgYKKwYBBAGC
# NwIBGwQIMAYBAQABAf8wDQYJKoZIhvcNAQELBQADggEBAGHSsbBauttLIctY2JeN
# Megumiz6/JeMPMfSZLEmyJABXYx2S0JisQ95yYUwubs+5+kCDDUnrh10LIyn8J96
# 2FhFC9bPCROYFhU+f83SelzEo4lXTR1XWsXyl5TSZLY77YBlR7UM5aH6Bc5tu/Kg
# 7nKAyOIoUJDxgWFrnzJpQPLtdhxI+WQGRAmakcVOosEdQclp/QtnCx6V1qC3Ne3i
# PHe2rsxfZ4bdbrqt3bYncY/1nyD12XVD7YR06beEMGUdi5eU6um3sQgjCFwQLZzO
# yXwey0XAwzgryTbZbGiE43qP7Bv1TgBsJcNzXCi9QsVqeYz+OvMMasizNI7CuzXH
# p0swggYKMIIE8qADAgECAhBSAOWqJVb8GobtlsnUSzPHMA0GCSqGSIb3DQEBBQUA
# MIHKMQswCQYDVQQGEwJVUzEXMBUGA1UEChMOVmVyaVNpZ24sIEluYy4xHzAdBgNV
# BAsTFlZlcmlTaWduIFRydXN0IE5ldHdvcmsxOjA4BgNVBAsTMShjKSAyMDA2IFZl
# cmlTaWduLCBJbmMuIC0gRm9yIGF1dGhvcml6ZWQgdXNlIG9ubHkxRTBDBgNVBAMT
# PFZlcmlTaWduIENsYXNzIDMgUHVibGljIFByaW1hcnkgQ2VydGlmaWNhdGlvbiBB
# dXRob3JpdHkgLSBHNTAeFw0xMDAyMDgwMDAwMDBaFw0yMDAyMDcyMzU5NTlaMIG0
# MQswCQYDVQQGEwJVUzEXMBUGA1UEChMOVmVyaVNpZ24sIEluYy4xHzAdBgNVBAsT
# FlZlcmlTaWduIFRydXN0IE5ldHdvcmsxOzA5BgNVBAsTMlRlcm1zIG9mIHVzZSBh
# dCBodHRwczovL3d3dy52ZXJpc2lnbi5jb20vcnBhIChjKTEwMS4wLAYDVQQDEyVW
# ZXJpU2lnbiBDbGFzcyAzIENvZGUgU2lnbmluZyAyMDEwIENBMIIBIjANBgkqhkiG
# 9w0BAQEFAAOCAQ8AMIIBCgKCAQEA9SNLXqXXirsy6dRX9+/kxyZ+rRmY/qidfZT2
# NmsQ13WBMH8EaH/LK3UezR0IjN9plKc3o5x7gOCZ4e43TV/OOxTuhtTQ9Sc1vCUL
# OKeMY50Xowilq7D7zWpigkzVIdob2fHjhDuKKk+FW5ABT8mndhB/JwN8vq5+fcHd
# +QW8G0icaefApDw8QQA+35blxeSUcdZVAccAJkpAPLWhJqkMp22AjpAle8+/Pxzr
# L5b65Yd3xrVWsno7VDBTG99iNP8e0fRakyiF5UwXTn5b/aSTmX/fze+kde/vFfZH
# 5/gZctguNBqmtKdMfr27Tww9V/Ew1qY2jtaAdtcZLqXNfjQtiQIDAQABo4IB/jCC
# AfowEgYDVR0TAQH/BAgwBgEB/wIBADBwBgNVHSAEaTBnMGUGC2CGSAGG+EUBBxcD
# MFYwKAYIKwYBBQUHAgEWHGh0dHBzOi8vd3d3LnZlcmlzaWduLmNvbS9jcHMwKgYI
# KwYBBQUHAgIwHhocaHR0cHM6Ly93d3cudmVyaXNpZ24uY29tL3JwYTAOBgNVHQ8B
# Af8EBAMCAQYwbQYIKwYBBQUHAQwEYTBfoV2gWzBZMFcwVRYJaW1hZ2UvZ2lmMCEw
# HzAHBgUrDgMCGgQUj+XTGoasjY5rw8+AatRIGCx7GS4wJRYjaHR0cDovL2xvZ28u
# dmVyaXNpZ24uY29tL3ZzbG9nby5naWYwNAYDVR0fBC0wKzApoCegJYYjaHR0cDov
# L2NybC52ZXJpc2lnbi5jb20vcGNhMy1nNS5jcmwwNAYIKwYBBQUHAQEEKDAmMCQG
# CCsGAQUFBzABhhhodHRwOi8vb2NzcC52ZXJpc2lnbi5jb20wHQYDVR0lBBYwFAYI
# KwYBBQUHAwIGCCsGAQUFBwMDMCgGA1UdEQQhMB+kHTAbMRkwFwYDVQQDExBWZXJp
# U2lnbk1QS0ktMi04MB0GA1UdDgQWBBTPmanqeyb0S8mOj9fwBSbv49KnnTAfBgNV
# HSMEGDAWgBR/02Wnwt3su/AwCfNDOfoCrzMxMzANBgkqhkiG9w0BAQUFAAOCAQEA
# ViLmNKTEYctIuQGtVqhkD9mMkcS7zAzlrXqgIn/fRzhKLWzRf3EafOxwqbHwT+QP
# DFP6FV7+dJhJJIWBJhyRFEewTGOMu6E01MZF6A2FJnMD0KmMZG3ccZLmRQVgFVlR
# OfxYFGv+1KTteWsIDEFy5zciBgm+I+k/RJoe6WGdzLGQXPw90o2sQj1lNtS0PUAo
# j5sQzyMmzEsgy5AfXYxMNMo82OU31m+lIL006ybZrg3nxZr3obQhkTNvhuhYuyV8
# dA5Y/nUbYz/OMXybjxuWnsVTdoRbnK2R+qztk7pdyCFTwoJTY68SDVCHERs9VFKW
# iiycPZIaCJoFLseTpUiR0zGCBHowggR2AgEBMIHJMIG0MQswCQYDVQQGEwJVUzEX
# MBUGA1UEChMOVmVyaVNpZ24sIEluYy4xHzAdBgNVBAsTFlZlcmlTaWduIFRydXN0
# IE5ldHdvcmsxOzA5BgNVBAsTMlRlcm1zIG9mIHVzZSBhdCBodHRwczovL3d3dy52
# ZXJpc2lnbi5jb20vcnBhIChjKTEwMS4wLAYDVQQDEyVWZXJpU2lnbiBDbGFzcyAz
# IENvZGUgU2lnbmluZyAyMDEwIENBAhBE7dMvNLbg1XINy7LQmAfFMAkGBSsOAwIa
```

```
# BQCgeDAYBgorBgEEAYI3AgEMMQowCKACgAChAoAAMBkGCSqGSIb3DQEJAzEMBgor
# BgEEAYI3AgEEMBwGCisGAQQBgjcCAQsxDjAMBgorBgEEAYI3AgEVMCMGCSqGSIb3
# DQEJBDEWBBRmOXOlNBgwTIZs+shp3SaVuAvjUjANBgkqhkiG9w0BAQEFAASCAQDk
# zIKzCVP4C0wQQcv1D7CAEcHhCJX9VLUVlENUKjafYM9eTqhSgpEFwgaAJAqR5XJV
# WiQzwpaG27B76/uSLlNA56OUeRtVZmjNmaTmUBx3jvquXlThgWo2KOAdQRVJc0b5
# NwIqFUe3DystwovhmUPj0erZQ2Z/giRvH+Km1Q6Vxoo76NO+0SaEtRbUVqTtaJ/Y
# 0E4Uan1vEmQ0pf8UEBqaQlkmg4umVSYESMqHImTubs0ecpY1lKs8+oNY8OVsn3wv
# Nf32ERP8HnApL5AXN8HTtguF0W5IYG0W4oU6HkGhTg5lNTFMQpkUJ3myLyAH5aBb
# 2uO3hhezJqJk/SjdTz4boYICCzCCAgcGCSqGSIb3DQEJBjGCAfgwggH0AgEBMHIw
# XjELMAkGA1UEBhMCVVMxHTAbBgNVBAoTFFN5bWFudGVjIENvcnBvcmF0aW9uMTAw
# LgYDVQQDEydTeW1hbnRlYyBUaW1lIFN0YW1waW5nIFNlcnZpY2VzIENBIC0gRzIC
# EA7P9DjI/r81bgTYapgbGlAwCQYFKw4DAhoFAKBdMBgGCSqGSIb3DQEJAzELBgkq
# hkiG9w0BBwEwHAYJKoZIhvcNAQkFMQ8XDTE1MDYxNjE0NDMzOFowIwYJKoZIhvcN
# AQkEMRYEFCPrcU08iLooi805klrLl/J9boEnMA0GCSqGSIb3DQEBAQUABIIBAEtm
# rlO04zf17KQkklo8EJRRynRRNX4uZJFDvypxX6+i+SSfHHbTOhRGzschqIpxzWTR
# u6ULda+LGC/AxaX/Y8YiWlIWVTI3ZG3qHeUawLVV1vBMBm3T3aGhbobZSLnHOlVM
# FgB6wrqlBy6Lydv5sLSKhKsISa4uVoobZUuQS7TGfW5bg+Z6zJE1Xe5wxhSmZZZx
# 09Q9pQfoHsL4slYPp6OB2vgvqlAk9GayYV7iFmmf8XYCEh2NCUuk7k2WbKE6r13m
# TVe5+NalODPJRQ7rKBcN2eXO/bjJzJQ5PwFQn4wGgxPCloAeFcg89dVnwZhtHBIW
# ixWKeKl+ktI+evwo3t8=
# SIG # End signature block
```

Install the Azure Connectivity Agent

The PowerShell script provided to configures and silently installs the Azure Connectivity Agent by supplying the PFX container file and the VPN profile parameters.

1. Open a **Windows PowerShell** with Administrator privileges on the Windows server VM.
2. Change to the directory you created containing the Network Access Client, PFX client certificate, and AzureConnectivityAgent.ps1 PowerShell script.
3. Start the installation of the Azure Connectivity Agent.
   ```
   C:\Azure>.\AzureConnectivityAgent.ps1 johndoe.pfx -PfxFilePassword
   mysecret -ProfileDescription ConnectivityAgent -ServerAddress
   vpn.acme.org -TunnelMode TCP -EncryptionAlgorithm AES256 -HashAlgorithm
   SHA1
   ```
4. The installation of the **Barracuda Network Access Client 3.6 x64** completes without the need for further user interaction.

Display All Azure Connectivity Agent Options

For a list of all available options, enter:

```
C:\Azure>.\Get-Help AzureConnectivityAgent.ps1 -full
```