

How to Configure the Azure Connectivity Agent

<https://campus.barracuda.com/doc/43847359/>

The Azure Connectivity Agent allows you to use the Barracuda NG 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.

In this article:

Before you Begin

- Deploy your Barracuda NG Firewall in the Azure Cloud.
- Deploy your Windows Server VMs in the same subnet(s) as the Barracuda NG 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.
 - 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 NG 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 Barracuda NG 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

.DESRIPTION
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      = '&quot;&amp;;description:&amp;&quot;;,
&quot;&amp;;server:&amp;&quot;;,
&quot;&amp;;mode:&amp;&quot;;,&quot;&amp;;encryption:&amp;&quot;;,
&quot;&amp;;hash:&amp;&quot;;,
&quot;&amp;;certname:&amp;&quot;;,&quot;&amp;;certserialnumber:&amp;&quot;;,
&quot;&amp;;certissuer:&amp;&quot;;, &quot;&amp;;certissuerX500:&amp;&quot;;
$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 '&quot;customer.inf&quot;;

$tunnelModeId      = 0
$encryptionAlgorithmId = 0
$hashAlgorithmId   = 0
$certName          = '&quot;&quot;;
$certSerial        = '&quot;&quot;;
$certIssuer        = '&quot;&quot;;
$certIssuerName    = '&quot;&quot;;

function AskTunnelMode()
{
    $result = 3
    $cl = New-Object System.Management.Automation.Host.ChoiceDescription

```

```

&quot;Reliability (&#amp;TCP)&quot;,,&quot;Reliability (TCP)&quot;;
    $c2 = New-Object System.Management.Automation.Host.ChoiceDescription
&quot;Response (&#amp;UDP)&quot;,,&quot;Response (UDP)&quot;;
    $c3 = New-Object System.Management.Automation.Host.ChoiceDescription
&quot;Optimized (&#amp;Hybrid)&quot;,,&quot;Optimized (Hybrid)&quot;;
    $choices =
[System.Management.Automation.Host.ChoiceDescription[]]($c1,$c2,$c3)
    $caption = &quot;Tunnel mode&quot;;
    $result = $Host.UI.PromptForChoice($caption, &quot;&quot;,, $choices, 2) #
'Hybrid' is default
    If($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 &quot;TCP&quot;){
        {
            return 1
        }
    ElseIf($str -eq &quot;UDP&quot;){
        {
            return 2
        }
    ElseIf($str -eq &quot;HYBRID&quot;){
        {
            return 3
        }
    Else
    {
        Throw &quot;Invalid Tunnel mode&quot;;
    }
}

function AskEncryptionAlgorithm()
{
    $result = 4
    $c1 = New-Object System.Management.Automation.Host.ChoiceDescription
&quot;&#amp;None&quot;,,&quot;No encryption&quot;;
    $c2 = New-Object System.Management.Automation.Host.ChoiceDescription
&quot;&#amp;AES&quot;,,&quot;AES&quot;;
    $c3 = New-Object System.Management.Automation.Host.ChoiceDescription
&quot;AES&#amp;256&quot;,,&quot;AES256&quot;;

```

```

    $c4 = New-Object System.Management.Automation.Host.ChoiceDescription
    &quot;&amp;Blowfish&quot;;&quot;Blowfish&quot;;
    $c5 = New-Object System.Management.Automation.Host.ChoiceDescription
    &quot;&amp;CAST&quot;;&quot;CAST&quot;;
    $c6 = New-Object System.Management.Automation.Host.ChoiceDescription
    &quot;&amp;DES&quot;;&quot;DES&quot;;
    $c7 = New-Object System.Management.Automation.Host.ChoiceDescription
    &quot;&amp;3DES&quot;;&quot;3DES&quot;;
    $choices =
[System.Management.Automation.Host.ChoiceDescription[]]($c1,$c2,$c3,$c4,$c5,$
c6,$c7)
    $caption = &quot;Encryption algorithm&quot;;
    $result = $Host.UI.PromptForChoice($caption, &quot;&quot;;, $choices, 1) #
'AES' is default
    If($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 &quot;NONE&quot;){
    {
        return 1
    }
    ElseIf($str -eq &quot;AES&quot;){
    {
        return 4
    }
    ElseIf($str -eq &quot;AES256&quot;){
    {
        return 64
    }
    ElseIf($str -eq &quot;BLOWFISH&quot;){
    {
        return 16
    }
    ElseIf($str -eq &quot;CAST&quot;){
    {
        return 8
    }

```

```
}
ElseIf($str -eq &quot;DES&quot;);)
{
    return 32
}
ElseIf($str -eq &quot;3DES&quot;);)
{
    return 2
}
Else
{
    Throw &quot;Invalid Encryption algorithm&quot;;
}
}

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

```
ElseIf($str -eq &quot;SHA1&quot;){
    {
        return 2
    }
Else
    {
        Throw &quot;Invalid Hash algorithm&quot;;
    }
}

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 &quot;=====&quot;;
    Write-Host &quot;Certificate information:&quot;;
    Write-Host &quot;=====&quot;;

    $subject = $cert.Subject
    Write-Host -NoNewLine &quot;Subject:          &quot;;
    Write-Host -fore Yellow $subject

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

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



```
Write-Host &quot;Please answer the following questions to complete
your configuration file:&quot;

If(!$ProfileDescription)
{
    Write-Host -NoNewLine &quot;Profile description: &quot;
    $script:ProfileDescription = Read-Host
    If(!$ProfileDescription)
    {
        $script:ProfileDescription = &quot;default&quot;
    }
}

If(!$ServerAddress)
{
    Write-Host -NoNewLine &quot;VPN server address/hostname: &quot;
    $script:ServerAddress = Read-Host
    If(!$ServerAddress)
    {
        $script:ServerAddress = &quot;vpnservers&quot;
    }
}

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 $infFullPath
Foreach($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 &quot;Using 'Import-PfxCertificate' for certificate
import...&quot;;
        If($PfxFilePassword)
        {
            $pfxSecureFilePassword = ConvertTo-SecureString $PfxFilePassword
-AsPlainText -Force
            Import-PfxCertificate $PfxFileName -CertStoreLocation
cert:\localMachine\my -Password $pfxSecureFilePassword
        }
        Else
        {
            Import-PfxCertificate $PfxFileName -CertStoreLocation
cert:\localMachine\my
        }
    }
    Else
    {
        Write-Host &quot;Using 'certutil' for certifcate import...&quot;;
        If($PfxFilePassword)
        {
            certutil -f -p $PfxFilePassword -importPFX $PfxFileName
        }
        Else
        {
            certutil -f -importPFX $PfxFileName
        }
    }
}
```

```
    }

    If($LASTEXITCODE -ne 0)
    {
        Throw &quot;&quot;;
    }
}

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

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

function RunConfigMode()
{
    Write-Host &quot;Entering configuration mode...&quot;;

    Try
    {
        ExtractCertificateData
        Try
        {
            GetProfileData
            Try
            {
                GenerateConfigFile
                Write-Host &quot;Configuration file has successfully been
generated.&quot;;
            }
            Catch
            {
                RemoveFileIfExists($infFullFilePath)
                Write-Error &quot;Configuration file could not be
generated!&quot;;
            }
        }
        Catch
    {
```

```
        Write-Error &quot;Profile data could not be retrieved!&quot;;
    }
}
Catch
{
    Write-Error &quot;Certificate data could not be extracted!&quot;;
}
}

function RunInstallMode()
{
    Write-Host &quot;Entering install mode...&quot;;

    If(!(IsConfigFileValid))
    {
        Write-Error &quot;Invalid configuration file!&quot;;
        return
    }

    Try
    {
        ImportCertificate
        Write-Host &quot;Certificate has successfully been imported.&quot;;

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

Write-Host
&quot;=====
=====&quot;;
Write-Host &quot;Welcome to the Barracuda Azure Connectivity Agent
Configuration and Setup Tool!&quot;;
Write-Host
&quot;=====
```

```

=====&quot;;
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+gYJKoZIhvcNAQcCoIIY6zCCG0cCAQExCzAJBgUrDgMCGGUAMGkGCisGAQQB
# gjcCAQSGWzBZMDQGCisGAQQBgjccAR4wJgIDAQAABBAfzDtgWUsITrck0sYpfvNR
# AgEAAgEAAgEAAgEAAgEAMCEwCQYFKw4DAhoFAAQUB+CXvjGrJJqKUJB+iqTELi58
# ldWgghPqMIID7jCCAlegAwIBAgIQfpPr+3zGTlnqS5p31Ab80zANBgkqhkiG9w0B
# AQUFADCBIzELMAkGA1UEBhMCWkExFTATBgNVBAGTDfDlc3Rlcm4gQ2FwZTEUMBIG
# A1UEBxMLRHVYyYmFudmlsbGUxDzANBgNVBAoTB1RoYXk0ZTEEdMBsGA1UECxMUVGhh
# d3RlIENlcnRpZmljYXRpb24xHzAdBgNVBAMTF1RoYXk0ZSBUaW1lc3RhbXBpbmVj
# Q0EwHhcNMTIxMjIxMDAwMDAwWhcNMjAxMjMwMjM1OTU5WjBeMQswCQYDVQQGEwJV
# UzEdMBsGA1UEChMUU3ltYW50ZWVwQ29ycG9yYXRpb24xMDAuBgNVBAMTJ1N5bWFu
# dGVjIFRpbWUgU3RhbXBpbmVjU2VydmljZXMgQ0EgLSBHMjCCASIwDQYJKoZIhvcN
# AQEBBQADggEPADCCAQoCggEBALGss0lUS5ccEgrYJXmRIlcqb9y4JJsRDc2vCvy5Q
# WvsUwna0QwElQ7Sh4kX06Ld7w3TMite0lAAC903tv7S3RCRrzV9F09FEzkMScxec
# i2m0K8uZHQxyGyZnCR+xMd37UWECU6aq9UksBXhFpS+JzueZ5/6M4lc/PcaS3Er4
# ezPkeQr78HWIQZz/xQNRmarXbJ+TaYdlKY0FwmAUxMjJ0xTawIHwHw103pIiq8r3
# +3R8J+b3Sht/p80eLa6K6qbmqicWfWH3mHERv0JQoUvLXfrlDqcsn6plINPYluji

```



```
# fKV0SET/GeJEB5IL12iEgF1qeGRFzWBgflTBE3zFefHJwXECAwEAAa0B+jCB9zAd
# BgNVHQ4EFgQUX5r1blzMzHSa1N197z/b7EyALt0wMgYIKwYBBQUHAQEEJjAkMCIG
# CCsGAQUFBzABhhZodHRwOi8vb2NzcC50aGF3dGUuY29tMBIGA1UdEwEB/wQIMAYB
# Af8CAQAwPwYDVR0fBDgwNjA0oDKgMIYuaHR0cDovL2Nybc50aGF3dGUuY29tL1Ro
# YXd0ZVRpbWVzdGFtcGluZ0NBLmNybdATBgNVHSUEDDAKBgg rBgEFBQcDCDA0BgNV
# HQ8BAf8EBAMCAQYwKAYDVR0RBCEwH6QdMBsXGTAXBgNVBAMTEFRpbWVtdGFtcC0y
# MDQ4LTEwDQYJKoZIhvcNAQEFBQADgYEAawbj3nvf1kwqu9otfrjCR27T4IGXTdf
# pLkFfo3qHJIJRG71betYfDDo+WmNI3MLEm9Hqa45EfgqsZuwGs0061mWAK30DE2y
# 0DGmCFwqevzieh1XTKhlG0L5QGIlm7HxzdqgyEijkhq3dLXPx13SYcqFgZepjhq
# IhKjURMdfrYwggSjMIIDi6ADAgECAhA0z/Q4yP6/NW4E2GqYGxpQMA0GCSqGSIb3
# DQEBBQUAMF4xCAzJBgNVBAYTAlVTMR0wGwYDVQQKEXRTEw1hbnRlYyBDb3Jw3Jh
# dGlvbjEwMC4GA1UEAxMnU3ltYW50ZWVtZGltZSBTdGFtcGluZyBTZXJ2aWwlc3Yy
# QSAAtIEcyMB4XDTEyMTAxODAwMDAwMFoXDTEyMTAxODAwMDAwMTIzNTk1OVowYj
# BhMCMVVMxHTAbBgNVBAoTFFN5bWVudGVjIENvcnBvcml0aW40aW9uMTQwMgYD
# VQDEYtT eW1hbnRlYyBUaWw1IFN0YW1waW5nIFNlc3ZyY2VzIFNpZ25lciAtIEc0
# MIIBIjAN # BgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAOmML0US4uy0nREm7Dv+h8GEKU50w
# mNutLA9Kxw7/hjxTVQ8VzgQ/K/2plpbZvmF5C1vJTIz25eBDSyKV7sIrQ8Gf2Gi0
# jkBP7oU4uRHFI/JkWPVMM90V6GuiKQC1yoezUvh3WPVF4kyw7BemVqonShQDhfu
# ltth00VRHc8SVguSR/yrrvZmPUeschLkudfzRC5xINKlBm9JYDh6NIipdC6Anqh
# d5NbZcPuF3S8QYYq3AhMjJKMkS2ed0QfaNaodHfbDlsyilaLM73ZY8hJnTrFxeoz
# C9Lxoxv0i77Zs1eL094Ep3oisiSuLsdwx50gyYI+wu9qU+ZCOEQKHkQzQIDAQAB
# o4IBVzCCAVMwDAYDVR0TAQH/BAIwADAwBgNVHSUBAf8EDDAKBgg rBgEFBQcDCDA0
# BgNVHQ8BAf8EBAMCB4AwcwYIKwYBBQUHAQEZZBlMCoGCCsGAQUFBzABhh5odHRw
# Oi8vdHMtb2NzcC53cy5zeW1hbnRlYy5jb20wNwYIKwYBBQUHMAKGG2h0dHA6Ly90
# cylhaWEud3Muc3ltYW50ZWMuY29tL3Rzcy1jYS1nMi5jZXIwPAYDVR0fBDUwMzAx
# oC+gLYYraHR0cDovL3RzLWVybC53cy5zeW1hbnRlYy5jb20vdHNzLWVhLWcyLmNy
# bDAoBgNVHREEITAFpB0wGzEZMBcGA1UEAxMQVGltdGVzVWVtZGltZSBTdGFtcGluZy
# HQ4EFgQURsZpow5KFB7VTNpSYxc/Xja8DeYwHwYDVR0jBBgwFoAUX5r1blzMzHSa
# 1N197z/b7EyALt0wDQYJKoZIhvcNAQEFBQADggEBAHg7tJEqAEzWj2IwN3ijhCch
# bxiy3iXcoNSUA6qGTiWfmkADHN3043nLIWgG2rYytG2/9CwmYzPkSWRtDebDZw73
# BaQ1bHyJFsbpst+y6d0gxnEPzZV03LZc3r03H0N45ni1zSgEIK0q8UvEiCmRDoDR
# EfzdXHZuT140RUZBbg2w6jiasTraCXEQ/Bx5tIB7rGn0/Zy2DBYr8X9bCT2bw+IW
# yh0BbQAu0A2oKY8s4bL0WqkBrxWcLC9JG9siu8P+eJRRw4axgohd8D20UaF5Mysu
# e7ncIAKtCetqGVvP6KUwVyyJST+5z3/Jvz4iaGNTmr1pdKzFHTx/kuDDvBzYBHUw
# ggU/MIIIEJ6ADAgECAhBE7dMvNLbg1XINy7LQmAfFMA0GCSqGSIb3DQEBcWUAMIG0
# MQswCQYDVOQGEwJVUzEXMBUGA1UEChM0VmVyaVNPZ2Z4IEluYy4xHzAdBgNVBAsT
# FlZlcm1TaWduIFRydXN0IE5ldHdvcmx0ZzA5BgNVBAsTMlRlcm1zIG9mIHVzZSBh
# dCBodHRwczovL3d3dy52ZXJpc2lnbi5jb20vcnBhIChjKTEwMS4wLWVhLWcyLmNy
# ZXJpU2lnbiBDbGFzcYAzIENvZGUuZ2lnbm1uZyAyMEwYDQYDQ0MTEwMTEwMTEwMTEw
# MDAwMFoXDTEyMTAxODAwMDAwMFoXDTEyMTAxODAwMDAwMTIzNTk1OVowcTElMAkGA1UE
# BhMCMVVMxHTAbBgNVBAoTFFN5bWVudGVjIENvcnBvcml0aW40aW9uMTQwMgYD
# VQDEYtT eW1hbnRlYyBUaWw1IFN0YW1waW5nIFNlc3ZyY2VzIFNpZ25lciAtIEc0
# MIIBIjAN # hkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAA6Va3PcbI+4j/Kx2xFX8Uqm/hj
# YTJtCWZ # 3JoNhgldIQujCsevSCuoXhsjiVlHIK6ljWF1im2BdoEeNXkQYG7WA1LU/BGNpVVA
# gfvaxaTj8L1bjpa7U4eiqmoSSQ4/u29cHB0CNStyeMviewN/varSHZgbB39hW3yX
# zfGIYGZvFRoUcn4XsuRVxsVMonzCA96G8G8cIpJ8a7z6lNwrTvhixS5mNqspXZq
```

```
# Ctm9Fzmv0W8p2XpeZpDGWdzsnh62DAKUe0TBBYw3rfdXD17jh9jR55t0xFvbBC+s
# jxIw2b/CPtnDP+LSN9FI6R+lmZsfnBo9SDT0oMGNZgpCvCrv4ACUXwIDAQAB04IB
# jTCCAYkwCQYDVR0TBAlwADA0BgNVHQ8BAf8EBAMCB4AwKwYDVR0fBCQwIjAgoB6g
# HIYaaHR0cDovL3NmLnN5bWNiLmNvbS9zZi5jcmwwZgYDVR0gBF8wXTBbBgtghkgB
# hvhFAQcXAzBMMCMGCCsGAQUFBwIBFhdodHRwczovL2Quc3ltY2IuY29tL2NwczAl
# BggrBgEFBQcCAjAZDBdodHRwczovL2Quc3ltY2IuY29tL3JwYTATBgNVHSUEDDAK
# BggrBgEFBQcDAzBxBggrBgEFBQcBAQRLMEkwHwYIKwYBBQUHMAGGE2h0dHA6Ly9z
# Zi5zeW1jZC5jb20wJgYIKwYBBQUHMAKGGmh0dHA6Ly9zZi5zeW1jYi5jb20vc2Yu
# Y3J0MB8GA1UdIwQYMBaAFM+Zqep7JvRlyY6P1/AFJu/j0qedMB0GA1UdDgQWBBS
# 3UFsXosnT2I3GwzEANOTjzWcszARBglghkgBhvhCAQEEBAMCBBAwFgYKKwYBBAGC
# NwIBGwQIMAYBAQABAf8wDQYJKoZIhvcNAQELBQADggEBAGHSsbBauttLIctY2JeN
# Megumiz6/JeMPmfSZLEmyJABXYx2S0JisQ95yYUwubs+5+kCDDUnrh10LIyn8J96
# 2FhFC9bPCROyFHu+f83SelzEo4lXTR1XwsXyl5TSZLY77YBlR7UM5aH6Bc5tu/Kg
# 7nKAY0IoUJDxgWFrnzJpQPLtdhxI+WQGRamakcV0osEdQclp/QtnCx6V1qC3Ne3i
# PHe2rsxfZ4bdbrrqt3bYncY/1nyD12XVD7YR06beEMGUdi5eU6um3sQgjCFwQLZz0
# yXwey0XAwzgryTbZbGiE43qP7Bv1TgBsJcNzXCi9QsVqeYz+0vMMasizNI7CuzXH
# p0swggYKMIIE8qADAgECAhBSA0WqJVb8GobtlSnUSzPHMA0GCSqGSIB3DQEBBQUA
# MIHKMQswCQYDVQQGEwJVUzEXMBUGA1UEChM0VmVyaVNPZ24sIEluYy4xHZAAdBgNV
# BAsTF1Zlcm1TaWduIFRydXN0IE5ldHdvcmsx0jA4BgNVBAsTMSShjKSAyMDA2IFZl
# cm1TaWduL2CBjbmMuIC0gRm9yIGF1dGhvcml6ZWQgdXNlIG9ubHkxRTBDBGNVBAMT
# PFZlcm1TaWduIENsYXNzIDMgUHVibGljIFByaW1hcnkgQ2VydGlnaWVhdGlvbiBB
# dXRob3JpdHkgLSBhNTAeFw0xMDAyMDgwMDAwMDBaFw0yMDAyMDcyMzU5NTlaMIG0
# MQswCQYDVQQGEwJVUzEXMBUGA1UEChM0VmVyaVNPZ24sIEluYy4xHZAAdBgNVBAsT
# FLZlcm1TaWduIFRydXN0IE5ldHdvcmsx0zA5BgNVBAsTMSlRlcm1zIG9mIHVzZSBh
# dCBodHRwczovL3d3dy52ZXJpc2lnbi5jb20vcnBhIChjKTEwMS4wLWVydVQDEyVW
# ZXJpU2lnbiBDbGFzcyAzIENvZGUgU2lnbm1uZyAyMDEwIENBIIBIjANBgkqhkiG
# 9w0BAQEFAA0CAQ8AMIIBCgKCAQEAA9SNLXqXXirsy6dRX9+/kxyZ+rRmY/qidfZT2
# NmsQ13WBMH8EaH/LK3UezR0Ijn9pLkC3o5x7g0CZ4e43TV/00xTuhtTQ9Sc1vCUL
# 0KeMY50Xowilq7D7zWpigkzVIidob2fHjhDuKKk+FW5ABT8mndhB/JwN8vq5+fcHd
# +QW8G0icaefApDw8QQA+35blxeSUcdZVaccAJkpAPLWhJqkMp22AjpaLe8+/Pxzr
# L5b65Yd3xrVwsno7VDBTG99iNP8e0fRakyiF5UwXTn5b/aSTmX/fze+kde/vFfZH
# 5/gZctguNBqmtKdMfr27Tww9V/EwlqY2jtaAdtcZLqXNfjQtiQIDAQAB04IB/jCC
# AfowEgYDVR0TAQH/BAgwBgEB/wIBADBwBgNVHSAEaTBnMGUGC2CGSAGG+EUBBxcD
# MFYwKAYIKwYBBQUHAQEWHGh0dHBz0i8vd3d3LnZlcm1zaWduLmNvbS9jchMwKgYI
# KwYBBQUHAgiwHhocaHR0cHM6Ly93d3d3cudmVyaVNPZ24uY29tL3JwYTA0BgNVHQ8B
# Af8EBAMCAQYwbQYIKwYBBQUHAQwEYTBfoV2gWzBZMFcwVRYJaW1hZ2UvZ2lmMCEw
# HzAHBgUrDgMCGGUj+XTGoasjY5rw8+AatRIGCx7GS4wJRYjaHR0cDovL2xvZ28u
# dmVyaVNPZ24uY29tL3ZzbG9nby5naWYwNAYDVR0fBC0wKzApoCegJYYjaHR0cDov
# L2Nybc52ZXJpc2lnbi5jb20vcGhMy1nNS5jcmwwNAYIKwYBBQUHAQEEDAmMCQG
# CCsGAQUFBzABhhodHRw0i8vb2Nzc52ZXJpc2lnbi5jb20wHQYDVR0lBBYwFAyI
# KwYBBQUHAwIGCCsGAQUFBwMDMCGGA1UdEQQhMB+kHTAbMRkwFwYDVQQDEExBZ290
# U2lnbk1QS0ktMi04MB0GA1UdDgQWBTPmanqeyb0S8m0j9fwBSbv49KnnTAFBgNV
# HSMEGDAWgBR/02Wnwt3su/AwCfND0foCrzMxMzANBgkqhkiG9w0BAQUFAA0CAQEA
# ViLmNKTEYctIuQgtVqhkD9mMkcS7zAzlrXqgIn/fRzhKLWzRf3Eaf0xwqbHwT+QP
# DFP6FV7+dJhJJIWBJhyRFEewTG0Mu6E01MZf6A2FJnMD0KmMZG3ccZLmRQVgFVlR
# 0fxYFGv+1KTtEwsIDEFy5zciBgm+I+k/RJoe6WGdzLGQXPw90o2sQj1lNtS0PUAo
```

```
# j5sQzyMmzEsgy5AfXYxMNM0820U31m+lIL006ybZrg3nxZr3obQhkTNvhuhYuyV8
# dA5Y/nUbYz/OMXybjxuWnsVTdoRbnK2R+qztk7pdyCFTwoJTY68SDVCHERS9VFKW
# iicyPZIaCJoFLseTpUiR0zGCBHowggR2AgEBMIHJMIG0MQswCQYDVQQGEwJVUzEX
# MBUGA1UEChMOVmVyaVNPZ24sIEluYy4xHzAdBgNVBAsTFmZlcm1TaWduIFRydXN0
# IE5ldHdvcmx0zA5BgNVBAsTMlRlcm1zIG9mIHVzZSBhdCBodHRwczovL3d3dy52
# ZXJpc2lnbi5jb20vcnBhIChjKTEwMS4wLAYDVQQDEyVWZXJpU2lnbiBDbGFzcyAz
# IENvZGUgU2lnbm1uZyAyMDEwIENBAhBE7dMvNLbg1XINy7LQmAfFMAkGBSs0AwIa
# BQCgeDAYBgorBgEEAYI3AgEMMQowCKACgAChAoAAMBkGCSqGSIb3DQEJAzEMBgor
# BgEEAYI3AgEEMBwGCisGAQQBgjcCAQsxDjAMBgorBgEEAYI3AgEVMCMGCSqGSIb3
# DQEJBDEWBBRm0X0lNBgwTIZs+shp3SaVuAvjUjANBgkqhkiG9w0BAQEFAASCAQDK
# zIKzCVP4C0wQQcv1D7CAEcHhCJX9VLUVlENUKjafYM9eTqhSgpEFwgaAJAqR5XJV
# WiQzwpag27B76/uSLlNA560UeRtVZmjNmaTmUBx3jvquXlThgWo2K0AdQRVJc0b5
# NwIqFUe3DystwovhmUPj0erZQ2Z/giRvH+Km1Q6Vxoo76N0+0SaEtRbUVqTtaJ/Y
# 0E4Uan1vEmQ0pf8UEBqaQlkmq4umVSYESmqHImTubs0ecpY1lKs8+oNY80Vsn3wv
# Nf32ERP8HnApL5AXN8HTtguF0W5IYG0W4oU6HkGhTg5lNTFMQpkUJ3myLyAH5aBb
# 2u03hhezJqJk/SjdTz4boYICczCCAgcGCSqGSIb3DQEJBJGCAfgwgGh0AgEBMHlw
# XjELMAkGA1UEBhMCVVMxHTAbBgNVBAoTFFN5bWVudGVjIENvcnBvcmlF0aW9uMTAw
# LgYDVQQDEydTeW1hbnRlYyBUaWw1lIFN0Yy1waW5nIFNlcnZpY2VzIENBIC0gRzIC
# EA7P9Dji/r81bgTYapgbGlaAwCQYFKw4DAhoFAKBdMBGCSqGSIb3DQEJAzELBgkq
# hkiG9w0BBwEwHAYJKoZIhvcNAQkFMQ8XDTE1MDYxNjE0NDMzOFowIwYJKoZIhvcN
# AQkEMRYEFCPrC0U08iLooi805klrLl/J9boEnMA0GCSqGSIb3DQEBAQUABIIBAEtm
# r1004zf17KQkklo8EJRRynRRNX4uZJFDvypxX6+i+SSfHHbT0hRGzsChqIpxzWTR
# u6ULda+LGC/AxaX/Y8YiWlIWVTI3ZG3qHeUawLVV1vBMBm3T3aGhbobZSLnH0lVM
# FgB6wrqlBy6Lydv5sLSKhKsISa4uVoobZUuQS7TGfW5bg+Z6zJE1Xe5wxhSmZZXx
# 09Q9pQfoHsL4sLYPp60B2vgvqlAk9GayYV7iFmmf8XYCEh2NCUuk7k2WbKE6r13m
# Tve5+Na10DPJRQ7rKBcN2eX0/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
```

© Barracuda Networks Inc., 2019 The information contained within this document is confidential and proprietary to Barracuda Networks Inc. No portion of this document may be copied, distributed, publicized or used for other than internal documentary purposes without the written consent of an official representative of Barracuda Networks Inc. All specifications are subject to change without notice. Barracuda Networks Inc. assumes no responsibility for any inaccuracies in this document. Barracuda Networks Inc. reserves the right to change, modify, transfer, or otherwise revise this publication without notice.