Sign in

Arno0x / PowerShellScripts Public

Notifications Fork 135 Star 441

Code Issues 2 Pull requests Actions Projects Security Insights

PowerShellScripts / proxyTunnel.ps1

147 lines (109 loc) · 4.98 KB

Code Blame Raw

```
<#
.SYNOPSIS

Creates a TCP Tunnel through the default system proxy. As such, it automatically handles proxy auth

Author: Arno0x0x (https://twitter.com/Arno0x0x)
License: GPL3
Required Dependencies: None
Optional Dependencies: None

.DESCRIPTION

Creates a TCP Tunnel through the default system proxy. As such, it automatically handles proxy auth

.PARAMETER bindIP

The local IP the tunnel will bind to. Defaults to 127.0.0.1 .

.PARAMETER bindPort

The local TCP port the tunnel will bind to. Defaults to 5555 .

.PARAMETER destHost

The destination host that should be reached through the tunnel.

```

```
.PARAMETER destPort

The destination port that should be reached through the tunnel.

.EXAMPLE

powershell .\proxyTunnel.ps1 -bindPort 4444 -destHost myserver.example.com -destPort 22

Then, an SSH connection to 127.0.0.1:4444 will be tunneled, through the corporate proxy, to myserve
#>

Param (
        [String]$bindIP = "127.0.0.1",

        [Int]$bindPort = 5555,

        [String]$destHost = $( Read-Host "Enter tunnel destination IP or Hostname: " ),

        [Int]$destPort = $( Read-Host "Enter tunnel destination port: " )
    )

$clientBuffer = new-object System.Byte[] 1024
$request = [System.Net.HttpWebRequest]::Create("http://" + $destHost + ":" + $destPort )
$request.Method = "CONNECT"

# Detect and set automatic proxy and network credentials
$proxy = [System.Net.WebRequest]::GetSystemWebProxy()
$proxy.Credentials = [System.Net.CredentialCache]::DefaultNetworkCredentials
$request.Proxy = $proxy

$listener = new-object System.Net.Sockets.TcpListener([System.Net.IPAddress]::Parse($bindIP), $binc


#-------------------------------------------------------------------------
# This script block is executed in a separate PowerShell object, as another
# thread. It reads data from the serverStream and writes it to the clientStream
# as long as there's data
$Script = {
        param($state)
        $serverBuffer = new-object System.Byte[] 1024

        $count = 0
        do {
                $count = $state.serverStream.Read($serverBuffer, 0 ,$serverBuffer.length)
                $state.clientStream.Write($serverBuffer, 0 , $count)
                $state.clientStream.Flush()
        } while ($count -gt 0)
}

#--------------------------------------------------------------------------
# Starting the TCP listener
$listener.start()

write-host "Waiting for a connection on port $bindPort..."
$client = $listener.AcceptTcpClient()
write-host "Connected from $($client.Client.RemoteEndPoint)"

#-----------------------------------------------------------------------------------------------
# Get the client side stream object to read/write to
$clientStream = $client.GetStream() # This is a System.Net.Sockets.NetworkStream

#-----------------------------------------------------------------------------------------------
# Get the server side response and corresponding stream object to read/write to
$serverResponse = $request.GetResponse()
$responseStream = $serverResponse.GetResponseStream()

#-----------------------------------------------------------------------------------------------
# Reflection inspection to retrieve and reuse the underlying networkStream instance
$BindingFlags= [Reflection.BindingFlags] "NonPublic,Instance"
$rsType = $responseStream.GetType()
$connectionProperty = $rsType.GetProperty("Connection", $BindingFlags)
$connection = $connectionProperty.GetValue($responseStream, $null)
$connectionType = $connection.GetType()
$networkStreamProperty = $connectionType.GetProperty("NetworkStream", $BindingFlags)
$serverStream = $networkStreamProperty.GetValue($connection, $null)

# This state object is used to pass various object by reference to the child PowerShell object (thr
# that is created afterwards
$state = [PSCustomObject]@{"serverStream"=$serverStream;"clientStream"=$clientStream}

# Create a child PowerShell object to run the background Socket receive method.
$PS = [PowerShell]::Create()
$PS.AddScript($Script).AddArgument($state) | Out-Null
[System.IAsyncResult]$AsyncJobResult = $null

try
{
        # The receive job is started asynchronously.
        $AsyncJobResult = $PS.BeginInvoke()

        do {
                $bytesReceived = $clientStream.Read($clientBuffer, 0, $clientBuffer.length)
                $bytesReceived = $clientStream.Read($clientBuffer, 0, $clientBuffer.length)
                $serverStream.Write($clientBuffer, 0 , $bytesReceived)
                #$text = [System.Text.Encoding]::ASCII.GetString($buffer, 0, $bytesReceived)
                #Write-Host $text

        } while ($client.Connected -or $clientStream.DataAvailable)
}
catch {
        $ErrorMessage = $_.Exception.Message
        Write-Host $ErrorMessage
}
finally {
        # Cleanup the client socket and child PowerShell process.
    if ($client -ne $null) {
        $client.Close()
        $client.Dispose()
        $client = $null
    }

        if ($listener -ne $null) {
                $listener.Stop()
        }

        write-host "Connection closed."

    if ($PS -ne $null -and $AsyncJobResult -ne $null) {
        $PS.EndInvoke($AsyncJobResult)
        $PS.Dispose()
    }
}
```