Sign in

danielbohannon / **Invoke-Obfuscation** Public

Notifications | Fork 766 | Star 3.7k

Code | Issues 11 | Pull requests 2 | Actions | Projects | Security | Insights

Invoke-Obfuscation / **Out-ObfuscatedStringCommand.ps1**

904 lines (704 loc) · 97.9 KB

Code | Blame | Raw

```
#   This file is part of Invoke-Obfuscation.
#
#   Copyright 2017 Daniel Bohannon <@danielhbohannon>
#         while at Mandiant <http://www.mandiant.com>
#
#   Licensed under the Apache License, Version 2.0 (the "License");
#   you may not use this file except in compliance with the License.
#   You may obtain a copy of the License at
#
#       http://www.apache.org/licenses/LICENSE-2.0
#
#   Unless required by applicable law or agreed to in writing, software
#   distributed under the License is distributed on an "AS IS" BASIS,
#   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
#   See the License for the specific language governing permissions and
#   limitations under the License.



Function Out-ObfuscatedStringCommand
{
<#
.SYNOPSIS

Master function that orchestrates the application of all string-based obfuscation functions to prov
```

```

Invoke-Obfuscation Function: Out-ObfuscatedStringCommand
Author: Daniel Bohannon (@danielhbohannon)
License: Apache License, Version 2.0
Required Dependencies: Out-EncapsulatedInvokeExpression (located in Out-ObfuscatedStringCommand.ps1
Optional Dependencies: None

.DESCRIPTION

Out-ObfuscatedStringCommand orchestrates the application of all string-based obfuscation functions
The available ObfuscationLevel/function mappings are:
1 --> Out-StringDelimitedAndConcatenated
2 --> Out-StringDelimitedConcatenatedAndReordered
3 --> Out-StringReversed

.PARAMETER ScriptBlock

Specifies a scriptblock containing your payload.

.PARAMETER Path

Specifies the path to your payload.

.PARAMETER ObfuscationLevel

(Optional) Specifies the obfuscation level for the given input PowerShell payload. If not defined t
The available ObfuscationLevel/function mappings are:
1 --> Out-StringDelimitedAndConcatenated
2 --> Out-StringDelimitedConcatenatedAndReordered
3 --> Out-StringReversed

.EXAMPLE

C:\PS> Out-ObfuscatedStringCommand {Write-Host 'Hello World!' -ForegroundColor Green; Write-Host 'O

IEX ((('Write-H'+'ost x'+'lcHello'+' Wor'+'ld!xlc -F'+'oregroundC'+'o'+'lor Gre'+'en'+'; Write-Host

C:\PS> Out-ObfuscatedStringCommand {Write-Host 'Hello World!' -ForegroundColor Green; Write-Host 'O

IEX( (("{17}{1}{6}{19}{14}{3}{5}{13}{16}{11}{20}{15}{10}{12}{2}{4}{8}{18}{7}{9}{0}" -f ' Green','-H

C:\PS> Out-ObfuscatedStringCommand {Write-Host 'Hello World!' -ForegroundColor Green; Write-Host 'O

$I4 ="noisserpxE-ekovnI|)93]rahC[]gnirtS[,'1Yp'(ecalpeR.)'ne'+'erG roloCd'+'nuo'+'rgero'+'F- 1'+'Y'

.NOTES
```

```

Out-ObfuscatedStringCommand orchestrates the application of all string-based obfuscation functions
This is a personal project developed by Daniel Bohannon while an employee at MANDIANT, A FireEye Co

.LINK

http://www.danielbohannon.com
#>

    [CmdletBinding( DefaultParameterSetName = 'FilePath')] Param (
        [Parameter(Position = 0, ValueFromPipeline = $True, ParameterSetName = 'ScriptBlock')]
        [ValidateNotNullOrEmpty()]
        [ScriptBlock]
        $ScriptBlock,

        [Parameter(Position = 0, ParameterSetName = 'FilePath')]
        [ValidateNotNullOrEmpty()]
        [String]
        $Path,

        [ValidateSet('1', '2', '3')]
        [Parameter(Position = 1)]
        [ValidateNotNullOrEmpty()]
        [Int]
        $ObfuscationLevel = (Get-Random -Input @(1..3)) # Default to random obfuscation level if $O
    )

    # Either convert ScriptBlock to a String or convert script at $Path to a String.
    If($PSBoundParameters['Path'])
    {
        Get-ChildItem $Path -ErrorAction Stop | Out-Null
        $ScriptString = [IO.File]::ReadAllText((Resolve-Path $Path))
    }
    Else
    {
        $ScriptString = [String]$ScriptBlock
    }

    # Set valid obfuscation levels for current token type.
    $ValidObfuscationLevels = @(0,1,2,3)

    # If invalid obfuscation level is passed to this function then default to highest obfuscation l
    If($ValidObfuscationLevels -NotContains $ObfuscationLevel) {$ObfuscationLevel = $ValidObfuscati

    Switch($ObfuscationLevel)
    {
    {
        0 {Continue}
```

```
HELPER FUNCTION :: Generates random syntax for invoking input PowerShell command.

Invoke-Obfuscation Function: Out-EncapsulatedInvokeExpression
Author: Daniel Bohannon (@danielhbohannon)
License: Apache License, Version 2.0
Required Dependencies: None
Optional Dependencies: None

.DESCRIPTION

Out-EncapsulatedInvokeExpression generates random syntax for invoking input PowerShell command. It

.PARAMETER ScriptString

Specifies the string containing your payload.

.EXAMPLE

C:\PS> Out-EncapsulatedInvokeExpression {Write-Host 'Hello World!' -ForegroundColor Green; Write-Ho
```

```
Write-Host 'Hello World!' -ForegroundColor Green; Write-Host 'Obfuscation Rocks!' -ForegroundColor

.NOTES

This cmdlet is most easily used by passing a script block or file path to a PowerShell script into
C:\PS> Out-ObfuscatedStringCommand {Write-Host 'Hello World!' -ForegroundColor Green; Write-Host 'O
C:\PS> Out-ObfuscatedStringCommand {Write-Host 'Hello World!' -ForegroundColor Green; Write-Host 'O
C:\PS> Out-ObfuscatedStringCommand {Write-Host 'Hello World!' -ForegroundColor Green; Write-Host 'O
This is a personal project developed by Daniel Bohannon while an employee at MANDIANT, A FireEye Co

.LINK

http://www.danielbohannon.com
#>

    [CmdletBinding()] Param (
        [Parameter(Position = 0)]
        [ValidateNotNullOrEmpty()]
        [String]
        $ScriptString
    )

    # The below code block is copy/pasted into almost every encoding function so they can maintain
    # Changes to below InvokeExpressionSyntax block should also be copied to those functions.
    # Generate random invoke operation syntax.
    $InvokeExpressionSyntax  = @()
    $InvokeExpressionSyntax += (Get-Random -Input @('IEX','Invoke-Expression'))
    # Added below slightly-randomized obfuscated ways to form the string 'iex' and then invoke it w
    # Though far from fully built out, these are included to highlight how IEX/Invoke-Expression is
    # These methods draw on common environment variable values and PowerShell Automatic Variable va
    $InvocationOperator = (Get-Random -Input @('.','&')) + ' '*(Get-Random -Input @(0,1))
    $InvokeExpressionSyntax += $InvocationOperator + "( `$ShellId[1]+`$ShellId[13]+'x')"
    $InvokeExpressionSyntax += $InvocationOperator + "( `$PSHome[" + (Get-Random -Input @(4,21)) +
    $InvokeExpressionSyntax += $InvocationOperator + "( `$env:ComSpec[4," + (Get-Random -Input @(15
    $InvokeExpressionSyntax += $InvocationOperator + "((" + (Get-Random -Input @('Get-Variable','GV
    $InvokeExpressionSyntax += $InvocationOperator + "( " + (Get-Random -Input @('$VerbosePreferenc
    # Commenting below option since $env:Public differs in string value for non-English operating s
    #$InvokeExpressionSyntax += $InvocationOperator + "( `$env:Public[13]+`$env:Public[5]+'x')"

    # Randomly choose from above invoke operation syntaxes.
    $InvokeExpression = (Get-Random -Input $InvokeExpressionSyntax)

    # Randomize the case of selected invoke operation.
    $InvokeExpression = Out-RandomCase $InvokeExpression

```

```
        # Choose random Invoke-Expression/IEX syntax and ordering: IEX ($ScriptString) or ($ScriptStrin
        $InvokeOptions  = @()
        $InvokeOptions += ' '*(Get-Random -Input @(0,1)) + $InvokeExpression + ' '*(Get-Random -Input @
        $InvokeOptions += ' '*(Get-Random -Input @(0,1)) + $ScriptString + ' '*(Get-Random -Input @(0,1

        $ScriptString = (Get-Random -Input $InvokeOptions)

        Return $ScriptString
    }
```