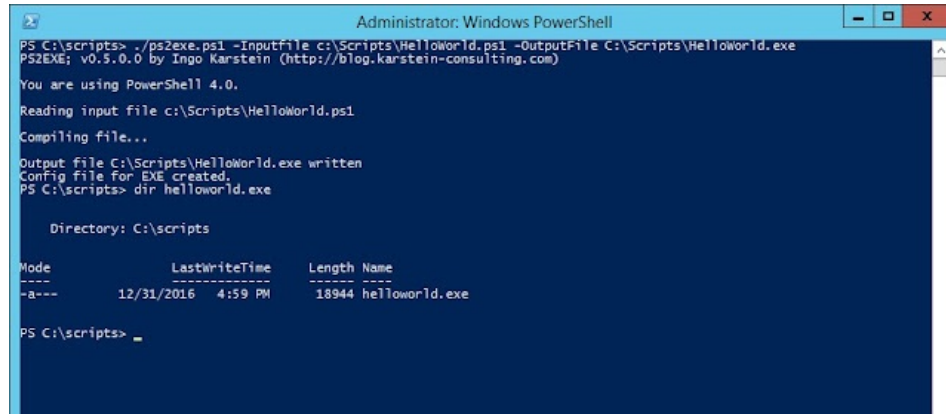


PS2EXE - Module To Compile Powershell Scripts To Executables

📅 3 DAYS AGO ⌚ 8:30 AM | POST SPONSORED BY FARADAYSEC | MULTIUSER PENTEST ENVIRONMENT

👤 ZION3R



Overworking of the great script of Ingo Karstein with GUI support. The GUI output and input is activated with one switch, real windows executables are generated. With Powershell 5.x support and graphical front end.
Module version.



You find the script based version here (<https://github.com/MScholtes/TechNet-Gallery>) and here: [PS2EXE-GUI: "Convert" PowerShell Scripts to EXE Files with GUI](#).

Author: Markus Scholtes

Version: 1.0.10

Date: 2021-04-10

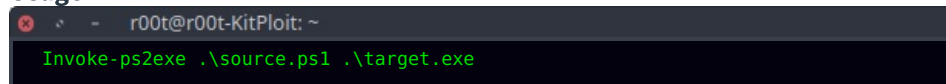
Installation



(on Powershell V4 you may have to install PowershellGet before) or download from here:

<https://www.powershellgallery.com/packages/ps2exe/>.

Usage



or



compiles "source.ps1" into the executable target.exe (if ".\target.exe" is omitted, output is written to ".\source.exe").

or start Win-PS2EXE for a graphical front end with



Parameter

FOLLOW US!



Your Email

Subscribe to our Newsletter

POPULAR



DNSTake - A Fast Tool To Check Missing Hosted DNS Zones That Can Lead To Subdomain Takeover

A fast tool to check missing hosted DNS zones that can lead to subdomain takeover. What is a DNS takeover? DNS takeover vulnerability...



On-The-Fly - Tool Which Gives Capabilities To Perform Pentesting Tests In Several Domains (IoT, ICS & IT)



Plutonium - Prototype Pollution Scanner Using Headless Chrome

Plutonium is a convenient way to scan at scale for pages that are vulnerable to client side prototype pollution via a URL payload. In t...



PS2EXE - Module To Compile Powershell Scripts To Executables

Overworking of the great script of Ingo Karstein with GUI support. The GUI output and input is activated with one switch, real windows exe...



CVE-2021-40444 PoC - Malicious docx generator to exploit CVE-2021-40444 (Microsoft Office Word Remote Code Execution)

Malicious docx generator to exploit CVE-2021-40444 (Microsoft Office Word Remote Code Execution)

```

r00t@r00t-KitPloit: ~
ps2exe [-inputFile] '<file_name>' [[-outputFile] '<file_name>'] [-prepareDebug]
[-x86|-x64] [-lcid <id>] [-STA|-MTA] [-noConsole] [-UNICODEEncoding]
[-credentialGUI] [-iconFile '<filename>'] [-title '<title>'] [-description '<description>']
[-company '<company>'] [-product '<product>'] [-copyright '<copyright>'] [-trademark '<trademark>']
[-version '<version>'] [-configFile] [-noOutput] [-noError] [-noVisualStyles]
[-requireAdmin]
[-supportOS] [-virtualize] [-longPaths]

```

debugging x86 or x64 = compile for 32-bit or 64-bit runtime only lcid = location ID for the compiled executable. Current user culture if not specified STA or MTA = 'Single Thread Apartment' or 'Multi Thread Apartment' mode noConsole = the resulting executable will be a Windows Forms app without a console window UNICODEEncoding = encode output as UNICODE in console mode credentialGUI = use GUI for prompting [credentials](#) in console mode iconFile = icon file name for the compiled executable title = title information (displayed in details tab of Windows Explorer's properties dialog) description = description information (not displayed, but embedded in executable) company = company information (not displayed, but embedded in executable) product = product information (displayed in details tab of Windows Explorer's properties dialog) copyright = copyright information (displayed in details tab of Windows Explorer's properties dialog) trademark = trademark information (displayed in details tab of Windows Explorer's properties dialog) version = version information (displayed in details tab of Windows Explorer's properties dialog) configFile = write config file (<outputfile>.exe.config) noOutput = the resulting executable will generate no standard output (includes verbose and information channel) noError = the resulting executable will generate no error output (includes warning and debug channel) noVisualStyles = disable visual styles for a generated windows GUI application (only with -noConsole) requireAdmin = if UAC is enabled, compiled executable run only in elevated context (UAC dialog appears if required) supportOS = use functions of newest Windows versions (execute [Environment]::OSVersion to see the difference) virtualize = application [virtualization](#) is activated (forcing x86 runtime) longPaths = enable long paths (> 260 characters) if enabled on OS (works only with Windows 10) ">

```

r00t@r00t-KitPloit: ~
    inputFile = Powershell script that you want to convert to executable (file has to be UTF8 or UTF16 encoded)
    outputFile = destination executable file name or folder, defaults to inputFile with extension '.exe'
    prepareDebug = create helpful information for debugging
    x86 or x64 = compile for 32-bit or 64-bit runtime only
    lcid = location ID for the compiled executable. Current user culture if not specified
    STA or MTA = 'Single Thread Apartment' or 'Multi Thread Apartment' mode
    noConsole = the resulting executable will be a Windows Forms app without a console window
    UNICODEEncoding = encode output as UNICODE in console mode
    credentialGUI = use GUI for prompting credentials in console mode
    iconFile = icon file name for the compiled executable
    title = title information (displayed in details tab of Windows Explorer's properties dialog)
    description = description information (not displayed, but embedded in executable)
    company = company information (not displayed, but embedded in executable)
    product = product information (displayed in details tab of Windows Explorer's properties dialog)
    copyright = copyright information (displayed in details tab of Windows Explorer's properties dialog)
    trademark = trademark information (displayed in details tab of Windows Explorer's properties dialog)
    version = version information (displayed in details tab of Windows Explorer's properties dialog)
    configFile = write config file (<outputfile>.exe.config)
    noOutput = the resulting executable will generate no standard output (includes verbose and information channel)
    noError = the resulting executable will generate no error output (includes warning and debug channel)
    noVisualStyles = disable visual styles for a generated windows GUI application (only with -noConsole)
    requireAdmin = if UAC is enabled, compiled executable run only in elevated context (UAC dialog appears if required)
    supportOS = use functions of newest Windows versions (execute [Environment]::OSVersion to see the difference)
    virtualize = application virtualization is activated (forcing x86 runtime)
    longPaths = enable long paths (> 260 characters) if enabled on OS (works only with Windows 10)

```

A generated executable has the following reserved parameters:

```

r00t@r00t-KitPloit: ~
-debug          Forces the executable to be debugged. It calls "System.Diagnostics.Debugger.Launch()".
-extract:<FILENAME> Extracts the powershell script inside the executable and saves it as FILENAME.
-               The script will not be executed.
-wait           At the end of the script execution it writes "Hit any key to exit..." and waits for a key to be pressed.
-end           All following options will be passed to the script inside the executable.
-               All preceding options are used by the executable itself and will not be passed to the script.

```

Remarks

List of cmdlets not implemented:

The basic input/output commands had to be rewritten in C# for PS2EXE. Not implemented are *Write-Progress* in console mode (too much work) and *Start-Transcript/Stop-Transcript* (no proper reference

implementation by Microsoft).

GUI mode output formatting:

Per default in powershell outputs of commandlets are formatted line per line (as an array of strings). When your command generates 10 lines of output and you use GUI output, 10 message boxes will appear each awaiting for an OK. To prevent this pipe your command to the commandlet Out-String. This will convert the output to one string array with 10 lines, all output will be shown in one message box (for example: `dir C:\ | Out-String`).

Config files:

PS2EXE can create config files with the name of the generated executable + ".config". In most cases those config files are not necessary, they are a manifest that tells which .Net Framework version should be used. As you will usually use the actual .Net Framework, try running your executable without the config file.

Parameter processing:

Compiled scripts process parameters like the original script does. One restriction comes from the Windows environment: for all executables all parameters have the type STRING, if there is no implicit conversion for your parameter type you have to convert explicitly in your script. You can even pipe content to the executable with the same restriction (all piped values have the type STRING).

Password security:

Never store passwords in your compiled script! One can simply decompile the script with the parameter -extract. For example



```
r00t@r00t-KitPloit: ~  
Output.exe -extract:C:\Output.ps1
```

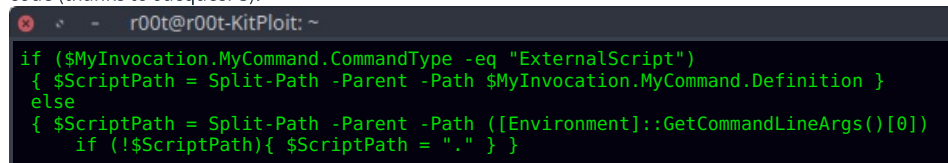
will decompile the script stored in Output.exe.

Script variables:

Since PS2EXE converts a script to an executable, script related variables are not available anymore. Especially the variable \$PSScriptRoot is empty.

The variable \$MyInvocation is set to other values than in a script.

You can retrieve the script/executable path independent of compiled/not compiled with the following code (thanks to JacquesFS):



```
r00t@r00t-KitPloit: ~  
if ($MyInvocation.MyCommand.CommandType -eq "ExternalScript")  
{ $ScriptPath = Split-Path -Parent -Path $MyInvocation.MyCommand.Definition }  
else  
{ $ScriptPath = Split-Path -Parent -Path ([Environment]::GetCommandLineArgs()[0])  
  if (!$ScriptPath){ $ScriptPath = "." } }
```

Window in background in -noConsole mode:

When an external window is opened in a script with -noConsole mode (i.e. for Get-Credential or for a command that needs a cmd.exe shell) the next window is opened in the background.

The reason for this is that on closing the external window windows tries to activate the parent window. Since the compiled script has no window, the parent window of the compiled script is activated instead, normally the window of Explorer or Powershell.

To work around this, \$Host.UI.RawUI.FlushInputBuffer() opens an invisible window that can be activated.

The following call of \$Host.UI.RawUI.FlushInputBuffer() closes this window (and so on).

The following example will not open a window in the background anymore as a single call of "ipconfig | Out-String" will do:



```
r00t@r00t-KitPloit: ~  
$Host.UI.RawUI.FlushInputBuffer()  
ipconfig | Out-String  
$Host.UI.RawUI.FlushInputBuffer()
```

Changes:

1.0.10 / 2021-04-10

- › parameter outputFile now accepts a target folder (without filename)

1.0.9 / 2021-02-28

- › new parameter UNICODEEncoding to output as UNICODE
- › changed parameter debug to prepareDebug
- › finally dared to use advanced parameters

1.0.8 / 2020-10-24

- › refactored

1.0.7 / 2020-08-21

- › bug fix for simultaneous progress bars in one pipeline

1.0.6 / 2020-08-10

- › prompt for choice behaves like Powershell now (console mode only)
- › (limited) support for Powershell Core (starts Windows Powershell in the background)
- › fixed processing of negative parameter values
- › support for animated progress bars (noConsole mode only)

1.0.5 / 2020-07-11

- › support for nested progress bars (noConsole mode only)

1.0.4 / 2020-04-19

- › Application.EnableVisualStyles() as default for GUI applications, new parameter - noVisualStyles to prevent this

1.0.3 / 2020-02-15

- › converted files from UTF-16 to UTF-8 to allow git diff
- › ignore control keys in secure string request in console mode

1.0.2 / 2020-01-08

- › added examples to github

1.0.1 / 2019-12-16

- › fixed "unlimited window width for GUI windows" issue in ps2exe.ps1 and Win-PS2EXE

1.0.0 / 2019-11-08

- › first stable module version

0.0.0 / 2019-09-15

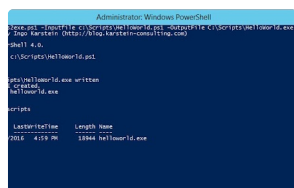
- › experimental

Download PS2EXE



TAGS

PS2EXE X UAC X VIRTUALIZATION X WINDOWS



PS2EXE - Module To Compile Powershell Scripts To Executables



QLOG - Windows Security Logging



Concealed Position - Bring Your Own Print Driver Privilege Escalation Tool

◀ PREVIOUS

InlineExecute-Assembly - A PoC Beacon Object File (BOF) That Allows Security Professionals To Perform In Process .NET Assembly Execution

NEXT ▶

CrowdSec - An Open-Source Massively Multiplayer Firewall Able To Analyze Visitor Behavior And Provide An Adapted Response To All Kinds Of Attacks

POST COMMENT

FACEBOOK

DISQUS

BLOG ARCHIVE

Blog Archive



RECOMMENDED

1. [SSD cloud server on DigitalOcean](#)
2. [Exploit Collector](#)
3. [BlackPloit](#)
4. [Hacking Reviews](#)
5. [Hacking Land](#)

SOCIAL



Your Email

Subscribe to our Newsletter



Blogger Feedburner