

# Forum:[Hack Forums](#)

Thread: [Dumping Powershell Variable Values in Runtime to Output File](#)

Hackers - April 28, 2024, 4:59 pm

Hi all, I want to share a simple script that can run a Powershell script & dump all of its variables to output.txt

This is useful because our script does NOT modify the original PS script, so it works with any. It also catches program crashes so you can get variables no matter when/where it exits!

Here is our wrapper.ps1 file that calls original.ps1 and catches all variables in runtime:

Code # Get the directory where this script is located

```
$scriptDir = Split-Path -Parent $MyInvocation.MyCommand.Path
```

```
# Define the path to the original script
```

```
$scriptPath = Join-Path -Path $scriptDir -ChildPath "original.ps1"
```

```
try {
```

```
    # Attempt to execute the original script
```

```
    . $scriptPath
```

```
} catch {
```

```
    # Handle errors from the original script (if it fails, crashes, or throws any error)
```

```
    Write-Host "An error occurred: $_"
```

```
} finally {
```

```
    # Capture all variables before the script exits, regardless of the outcome
```

```
    Get-Variable | Where-Object {
```

```
        $_.Name -notin @('scriptDir', 'scriptPath', 'HOME', 'PWD', 'PSVersionTable', 'Host', 'true', 'false', 'null', 'Error', 'LASTEXITCODE')
```

```
    } | ForEach-Object {
```

```
        "$($_.Name) = $($_.Value)" | Out-File "output.txt" -Append
```

```
    }
```

```
    # Optionally, log the completion of variable capture
```

```
    Write-Host "Variable capture complete."
```

```
}
```

```
# Check if the last exit code signals an error (from external programs or the script itself if it sets $LASTEXITCODE)
```

```
if ($LASTEXITCODE -ne 0) {
```

```
    Write-Host "The script exited with an error code: $LASTEXITCODE"
```

```
}
```

For testing, here is original.ps1 which I used:

Code # original.ps1 - Example PowerShell script with various operations and intentional error

```
# Define some variables
```

```
$var1 = "Hello World"
```

```
$var2 = 25
```

```
$var3 = @{"Name" = "Sample"; "Value" = 42 }
```

```
# Print variable values
Write-Host "var1: $var1"
Write-Host "var2: $var2"
Write-Host "var3: $($var3.Name), $($var3.Value)"

# Perform a calculation
$result = $var2 * 3
Write-Host "Result of calculation: $result"

# Introduce an error
Write-Host "Attempting an operation that will fail..."
$failedOperation = 1 / 0 # This division by zero will trigger an exception
```

# If the script reaches here, it means the error was not fatal  
 Write-Host "This line will not be executed because of the preceding error."  
 To bring it all together, you could put the original.ps1 code into wrapper.ps1, and you could then package wrapper.ps1 into a C++ program or similar. It's up to you! I was helping out another user and they are going to modify this code for another use case & said I could share this. I hope this helps some others here too, but remember this is just a POC on variable dumping and is not a "final program" for anything specific. Use at your own risk, for educational & personal & legal uses only.

Here is an example log from command prompt / cmd.exe :

```
Quote:var1: Hello World
var2: 25
var3: Sample, 42
Result of calculation: 75
Attempting an operation that will fail...
An error occurred: Attempted to divide by zero.
Variable capture complete.
The script exited with an error code:
PowerShell script executed and variables captured.
Here is the associated output.txt from the runtime dump:
Quote:$ =
? = True
^ =
_ =
args =
ConfirmPreference = High
ConsoleFileName =
DebugPreference = SilentlyContinue
ErrorActionPreference = Continue
ErrorView = NormalView
ExecutionContext = System.Management.Automation.EngineIntrinsics
FormatEnumerationLimit = 4
InformationPreference = SilentlyContinue
input =
MaximumAliasCount = 4096
MaximumDriveCount = 4096
MaximumErrorCount = 256
MaximumFunctionCount = 4096
MaximumHistoryCount = 4096
MaximumVariableCount = 4096
MyInvocation = System.Management.Automation.InvocationInfo
```

NestedPromptLevel = 0  
OutputEncoding = System.Text.ASCIIEncoding  
PID = 49196  
PROFILE = C:\Users\nodes\Documents\WindowsPowerShell\Microsoft.PowerShell\_profile.ps1  
ProgressPreference = Continue  
PSBoundParameters = System.Management.Automation.PSBoundParametersDictionary  
PSCommandPath = C:\Users\nodes\Desktop\Personal\C++\DumpPS\x64\Debug\original.ps1  
PSCulture = en-US  
PSDefaultParameterValues = System.Management.Automation.DefaultParameterDictionary  
PSEdition = Desktop  
PSEmailServer =  
PSHOME = C:\Windows\System32\WindowsPowerShell\v1.0  
PSItem =  
PSScriptRoot = C:\Users\nodes\Desktop\Personal\C++\DumpPS\x64\Debug  
PSSessionApplicationName = wsman  
PSSessionConfigurationName = <http://schemas.microsoft.com/powershell/Microsoft.PowerShell>  
PSSessionOption = System.Management.Automation.Remoting.PSSessionOption  
PSUICulture = en-US  
result = 75  
ShellId = Microsoft.PowerShell  
StackTrace = at  
System.Management.Automation.ExceptionHandlingOps.CheckActionPreference(FunctionContext funcContext, Exception exception)  
at System.Management.Automation.Interpreter.ActionCallInstruction`2.Run(InterpretedFrame frame)  
at  
System.Management.Automation.Interpreter.EnterTryCatchFinallyInstruction.Run(InterpretedFrame frame)  
at  
System.Management.Automation.Interpreter.EnterTryCatchFinallyInstruction.Run(InterpretedFrame frame)  
at System.Management.Automation.Interpreter.Interpreter.Run(InterpretedFrame frame)  
at System.Management.Automation.Interpreter.LightLambda.RunVoid1[T0](T0 arg0)  
at System.Management.Automation.DlrScriptCommandProcessor.RunClause(Action`1 clause, Object dollarUnderbar, Object inputToProcess)  
--- End of stack trace from previous location where exception was thrown ---  
at System.Runtime.ExceptionServices.ExceptionDispatchInfo.Throw()  
at  
System.Management.Automation.Internal.PipelineProcessor.SynchronousExecuteEnumerate(Object input)  
at System.Management.Automation.PipelineOps.InvokePipeline(Object input, Boolean ignoreInput, CommandParameterInternal[] pipeElements, CommandBaseAst[] pipeElementAsts, CommandRedirection[] commandRedirections, FunctionContext funcContext)  
at System.Management.Automation.Interpreter.ActionCallInstruction`6.Run(InterpretedFrame frame)  
at  
System.Management.Automation.Interpreter.EnterTryCatchFinallyInstruction.Run(InterpretedFrame frame)  
var1 = Hello World  
var2 = 25  
var3 = System.Collections.Hashtable  
VerbosePreference = SilentlyContinue  
WarningPreference = Continue

WhatIfPreference = False

**InspectorGadget** - April 28, 2024, 5:09 pm

Nice work mate, likely useful for the malware researchers out there!

**Hackers** - April 28, 2024, 5:14 pm

(Apr 28, 2024 05:09 PM)InspectorGadget Wrote: [\[CITE\]](#) Nice work mate, likely useful for the malware researchers out there!

I hope so! Going to be releasing a few blue team things to help detect, analyze, etc over time.

Really busy but hoping to provide more of these tools

**nucleo\_type09** - April 29, 2024, 5:49 am

nice one nodes, u know im not really a windows guy, but i never understood or never had the chance to research it the difference between powershell and cmd, r they related somehow or PS is a newer more advanced version of CMD? the only difference that jumps into my mind atm is that on is blue the other is black :)

**Hackers** - April 29, 2024, 6:06 am

(Apr 29, 2024 05:49 AM)nucleo\_type09 Wrote: [\[CITE\]](#) nice one nodes, u know im not really a windows guy, but i never understood or never had the chance to research it the difference between powershell and cmd, r they related somehow or PS is a newer more advanced version of CMD? the only difference that jumps into my mind atm is that on is blue the other is black :)

PS is definitely newer and used for a lot, and it is more advanced. AFAIK there really isn't anything you can't do in Powershell that you can do in cmd, so most people made the swap to PS. I've heard it's also easier to manage admin-privilege related features.

**nucleo\_type09** - April 29, 2024, 6:10 am

(Apr 29, 2024 06:06 AM)nodes Wrote: [\[CITE\]](#) PS is definitely newer and used for a lot, and it is more advanced. AFAIK there really isn't anything you can't do in Powershell that you can do in cmd, so most people made the swap to PS. I've heard it's also easier to manage admin-privilege related features.

oh, now i see what's the difference :), thanks for making this clear for me nodes

**Hackers** - April 29, 2024, 6:35 am

(Apr 29, 2024 06:10 AM)nucleo\_type09 Wrote: [\[CITE\]](#) oh, now i see what's the difference :), thanks for making this clear for me nodes

No problem!!

----

Just re-reading the output.txt and realizing that var3 isn't actually getting the final value due to how it's setup. I might fix this but am lacking time, so just be aware you might want to adjust things to catch that use case if you use these scripts guys!

**BlackWidow1** - April 29, 2024, 6:42 am

Great job, man. I'm gonna try to use this.

**nucleo\_type09** - April 29, 2024, 7:03 am

(Apr 29, 2024 06:35 AM)nodes Wrote: [\[CITE\]](#) No problem!!

----

Just re-reading the output.txt and realizing that var3 isn't actually getting the final value due to how

it's setup. I might fix this but am lacking time, so just be aware you might want to adjust things to catch that use case if you use these scripts guys!

thanks man, i like how you do have family, a job, and still you're able to provide such things to the community, thanks :)

Hackers - April 29, 2024, 2:30 pm

(Apr 29, 2024 06:42 AM)BlackWidow1 Wrote: [\[CITE\]](#) Great job, man. I'm gonna try to use this. Awesome, let me know if you find any bugs!!

(Apr 29, 2024 07:03 AM)nucleo\_type09 Wrote: [\[CITE\]](#) thanks man, i like how you do have family, a job, and still you're able to provide such things to the community, thanks :)

No problem haha! it's just as much for me as it is for you  
it's my portfolio, experience, and skills that benefit the most

nucleo\_type09 - April 29, 2024, 3:03 pm

(Apr 29, 2024 02:30 PM)nodes Wrote: [\[CITE\]](#) Awesome, let me know if you find any bugs!!

No problem haha! it's just as much for me as it is for you  
it's my portfolio, experience, and skills that benefit the most  
congrats man, i forgot to ask you, do u also upload what u share here on github?

Hackers - April 30, 2024, 9:03 pm

(Apr 29, 2024 03:03 PM)nucleo\_type09 Wrote: [\[CITE\]](#) congrats man, i forgot to ask you, do u also upload what u share here on github?

yes, depending on what it is I do!

mordock1998 - May 1, 2024, 9:44 am

(Apr 28, 2024 04:59 PM)nodes Wrote: [\[CITE\]](#) Hi all, I want to share a simple script that can run a Powershell script & dump all of its variables to output.txt

This is useful because our script does NOT modify the original PS script, so it works with any. It also catches program crashes so you can get variables no matter when/where it exits!

Here is our wrapper.ps1 file that calls original.ps1 and catches all variables in runtime:

Code # Get the directory where this script is located

```
$scriptDir = Split-Path -Parent $MyInvocation.MyCommand.Path
```

```
# Define the path to the original script
```

```
$scriptPath = Join-Path -Path $scriptDir -ChildPath "original.ps1"
```

```
try {
```

```
    # Attempt to execute the original script
```

```
    . $scriptPath
```

```
} catch {
```

```
    # Handle errors from the original script (if it fails, crashes, or throws any error)
```

```
    Write-Host "An error occurred: $_"
```

```
} finally {
```

```
    # Capture all variables before the script exits, regardless of the outcome
```

```
    Get-Variable | Where-Object {
```

```
        $_.Name -notin @( 'scriptDir', 'scriptPath', 'HOME', 'PWD', 'PSVersionTable', 'Host', 'true',  
'false', 'null', 'Error', 'LASTEXITCODE' )
```

```
    } | ForEach-Object {
```

```
        "$($_.Name) = $($_.Value)" | Out-File "output.txt" -Append
```

```

}
# Optionally, log the completion of variable capture
Write-Host "Variable capture complete."
}

# Check if the last exit code signals an error (from external programs or the script itself if it sets $LASTEXITCODE)
if ($LASTEXITCODE -ne 0) {
    Write-Host "The script exited with an error code: $LASTEXITCODE"
}

```

For testing, here is original.ps1 which I used:

Code # original.ps1 - Example PowerShell script with various operations and intentional error

```

# Define some variables
$var1 = "Hello World"
$var2 = 25
$var3 = @{ Name = "Sample"; Value = 42 }

# Print variable values
Write-Host "var1: $var1"
Write-Host "var2: $var2"
Write-Host "var3: $($var3.Name), $($var3.Value)"

# Perform a calculation
$result = $var2 * 3
Write-Host "Result of calculation: $result"

# Introduce an error
Write-Host "Attempting an operation that will fail..."
$failedOperation = 1 / 0 # This division by zero will trigger an exception

# If the script reaches here, it means the error was not fatal
Write-Host "This line will not be executed because of the preceding error."

```

To bring it all together, you could put the original.ps1 code into wrapper.ps1, and you could then package wrapper.ps1 into a C++ program or similar. It's up to you! I was helping out another user and they are going to modify this code for another use case & said I could share this. I hope this helps some others here too, but remember this is just a POC on variable dumping and is not a "final program" for anything specific. Use at your own risk, for educational & personal & legal uses only.

Here is an example log from command prompt / cmd.exe :

Here is the associated output.txt from the runtime dump:

cool, code looks good, good structure, but i would address some points:

While it had a try-catch block, it only printed the error message without providing additional context, The variable capture logic in the finally block was simplistic even tho acceptable :), the output file path was hardcoded, which could cause issues if the script is executed from a different directory, also... it didn't specify the encoding for the output file, that could lead to encoding-related issues. There was also no logging mechanism, hey, have a look at this

Code try {

```

# Get the directory where this script is located
$scriptDir = Split-Path -Parent $MyInvocation.MyCommand.Path

# Define the path to the original script
$scriptPath = Join-Path -Path $scriptDir -ChildPath "original.ps1"

# Attempt to execute the original script
. $scriptPath

# If the script executes successfully, log the completion
Write-Host "The script executed successfully."
} catch {
    # Handle errors from the original script (if it fails, crashes, or throws any error)
    Write-Host "An error occurred: $_"
} finally {
    # Capture all variables before the script exits, regardless of the outcome
    $excludedVariables = @('scriptDir', 'scriptPath', 'HOME', 'PWD', 'PSVersionTable', 'Host', 'true',
'false', 'null', 'Error', 'LASTEXITCODE')
    $variableCapture = Get-Variable | Where-Object { $_.Name -notin $excludedVariables } |
ForEach-Object { "$($_.Name) = $($_.Value)" }

    # Write the captured variables to a file
    $outputFile = Join-Path -Path $scriptDir -ChildPath "output.txt"
    $variableCapture | Out-File -FilePath $outputFile -Append -Encoding utf8

    # Optionally, log the completion of variable capture
    Write-Host "Variable capture complete."
}

# Check if the last exit code signals an error (from external programs or the script itself if it sets
$LASTEXITCODE)
if ($LASTEXITCODE -ne 0) {
    Write-Host "The script exited with an error code: $LASTEXITCODE"
}

```

**Qismon** - June 4, 2024, 6:39 am

Nice script. I made a similar one before but not runtime, just as a script import if you want to use parts of it go for it. Prints out all user defined global and normal variables as well as functions.

```

Code function Get-ScriptVariables() {
    Param(
        [Parameter(Mandatory=$True, Position=0)]
        [String] $poshSource
    )

    $reservedVariables = @(
        'ErrorActionPreference',
        'FormatEnumerationLimit',
        'MaximumAliasCount',
        'MaximumDriveCount',
        'MaximumErrorCount',
        'MaximumFunctionCount',
        'MaximumHistoryCount',

```

```

    'MaximumVariableCount',
    'PROFILE',
    'PSSessionOption',
    'global',
    'env',
    ''
    -
)

```

```

$specialVariables =
[PsObject].Assembly.GetType('System.Management.Automation.SpecialVariables').GetFields('Non
Public,Static')

```

```

$reservedVariables += ($specialVariables | Where-Object FieldType -eq ([string]) | Select-Object
-Expand Name)

```

```

$reservedVariables = $reservedVariables | ForEach-Object {'$' + $_}
$reservedAliases = @(Get-Alias * | Select-Object -Expand Name)

```

```

$variablesRegex = '$\w+(?!.*:)'
$globalRegex = '(?i)\$bglobal\b:\w+'
$functionsRegex = '(?i)\bfunction\b[ ]\w+|- \w+|\bfunction\b[ ]\w+'

```

```

$globalScriptVariables = Select-String -Pattern $globalRegex -Input $poshSource -AllMatches |
foreach {$_.Matches} | Select-Object -Unique

```

```

$UserDefinedVariables = Select-String -Pattern $variablesRegex -Input $poshSource
-AllMatches | foreach {$_.Matches} | Where-Object {$_ -notin $reservedVariables}
$UserDefinedVariables = $UserDefinedVariables | Select-Object -Unique

```

```

$UserDefinedFunctions = Select-String -Pattern $functionsRegex -Input $poshSource
-AllMatches | foreach {$_.Matches} | Where-Object {$_ -notin $reservedVariables}
$UserDefinedFunctions = $UserDefinedFunctions | ForEach-Object {$_ -iReplace('function ', '')} |
Select-Object -Unique

```

Clear-Host

```

Write-Host "[GLOBAL VARIABLES]"`n"
foreach($globalVariable in $globalScriptVariables) {
    Write-Host $globalVariable
}
Write-Host ('-'*42)
Write-Host "`n"

```

```

Write-Host "[USER VARIABLES]"`n"
foreach($userVariable in $userDefinedVariables) {
    Write-Host $userVariable
}
Write-Host ('-'*42)
Write-Host "`n"

```

```

Write-Host "[USER FUNCTIONS]"`n"
foreach($userFunction in $userDefinedFunctions) {
    Write-Host $userFunction
}
}

```

```
function Import-PoshSource() {
    Param(
        [Parameter(Position=0, Mandatory=$True)]
        [string] $poshFilePath
    )

    $poshSource = [IO.File]::ReadAllText($poshFilePath)

    try {
        $poshSource = [ScriptBlock]::Create($poshSource)
    }
    catch{}

    return $poshSource
}
```

```
$poshFilePath = 'C:\Users\dev\Desktop\file.ps1'
$poshSource = Import-PoshSource $poshFilePath
```

```
Get-ScriptVariables $poshSource
```

**Ren And Stimpy** - June 4, 2024, 6:41 am

(Jun 4, 2024 06:39 AM)Qismon Wrote: [CITE](#) Nice script. I made a similar one before but not runtime, just as a script import if you want to use parts of it go for it. Prints out all user defined global and normal variables as well as functions.

```
Code function Get-ScriptVariables() {
    Param(
        [Parameter(Mandatory=$True, Position=0)]
        [String] $poshSource
    )

    $reservedVariables = @(
        'ErrorActionPreference',
        'FormatEnumerationLimit',
        'MaximumAliasCount',
        'MaximumDriveCount',
        'MaximumErrorCount',
        'MaximumFunctionCount',
        'MaximumHistoryCount',
        'MaximumVariableCount',
        'PROFILE',
        'PSSessionOption',
        'global',
        'env',
        ' '
    )

    $specialVariables =
[PsObject].Assembly.GetType('System.Management.Automation.SpecialVariables').GetFields('Non
```

```
Public,Static')
```

```
$reservedVariables += ($specialVariables | Where-Object FieldType -eq ([string]) | Select-Object  
-Expand Name)
```

```
$reservedVariables = $reservedVariables | ForEach-Object {('$' + $_)}
```

```
$reservedAliases = @(Get-Alias * | Select-Object -Expand Name)
```

```
$variablesRegex = '$\w+(?!.*:).'
```

```
$globalRegex = '(?i)\$bglobal\b:\w+'
```

```
$functionsRegex = '(?i)\bfunction\b[ ]\w+\-\w+|\bfunction\b[ ]\w+'
```

```
$globalScriptVariables = Select-String -Pattern $globalRegex -Input $poshSource -AllMatches |  
foreach {$_ .Matches} | Select-Object -Unique
```

```
$userDefinedVariables = Select-String -Pattern $variablesRegex -Input $poshSource  
-AllMatches | foreach {$_ .Matches} | Where-Object {$_ -notin $reservedVariables}
```

```
$userDefinedVariables = $userDefinedVariables | Select-Object -Unique
```

```
$userDefinedFunctions = Select-String -Pattern $functionsRegex -Input $poshSource  
-AllMatches | foreach {$_ .Matches} | Where-Object {$_ -notin $reservedVariables}
```

```
$userDefinedFunctions = $userDefinedFunctions | ForEach-Object {$_ -iReplace('function ', '')} |  
Select-Object -Unique
```

```
Clear-Host
```

```
Write-Host "[GLOBAL VARIABLES]`n"
```

```
foreach($globalVariable in $globalScriptVariables) {
```

```
    Write-Host $globalVariable
```

```
}
```

```
Write-Host ('-*42)
```

```
Write-Host "`n"
```

```
Write-Host "[USER VARIABLES]`n"
```

```
foreach($userVariable in $userDefinedVariables) {
```

```
    Write-Host $userVariable
```

```
}
```

```
Write-Host ('-*42)
```

```
Write-Host "`n"
```

```
Write-Host "[USER FUNCTIONS]`n"
```

```
foreach($userFunction in $userDefinedFunctions) {
```

```
    Write-Host $userFunction
```

```
}
```

```
}
```

```
function Import-PoshSource() {
```

```
    Param(
```

```
        [Parameter(Position=0, Mandatory=$True)]
```

```
        [string] $poshFilePath
```

```
)
```

```
$poshSource = [IO.File]::ReadAllText($poshFilePath)
```

```

try {
    $poshSource = [ScriptBlock]::Create($poshSource)
}
catch{}

return $poshSource
}

```

```

$poshFilePath = 'C:\Users\dev\Desktop\file.ps1'
$poshSource = Import-PoshSource $poshFilePath

```

Get-ScriptVariables \$poshSource  
 Nice share mate!! Always giving back the community!

**Hackers** - November 19, 2024, 12:51 pm

(Jun 4, 2024 06:39 AM)Qismon Wrote: [\[CITE\]](#) Nice script. I made a similar one before but not runtime, just as a script import if you want to use parts of it go for it. Prints out all user defined global and normal variables as well as functions.

```

Code function Get-ScriptVariables() {
    Param(
        [Parameter(Mandatory=$True, Position=0)]
        [String] $poshSource
    )

    $reservedVariables = @(
        'ErrorActionPreference',
        'FormatEnumerationLimit',
        'MaximumAliasCount',
        'MaximumDriveCount',
        'MaximumErrorCount',
        'MaximumFunctionCount',
        'MaximumHistoryCount',
        'MaximumVariableCount',
        'PROFILE',
        'PSSessionOption',
        'global',
        'env',
        ''
    )

    $specialVariables =
    [PsObject].Assembly.GetType('System.Management.Automation.SpecialVariables').GetFields('Non
    Public,Static')
    $reservedVariables += ($specialVariables | Where-Object FieldType -eq ([string]) | Select-Object
    -Expand Name)
    $reservedVariables = $reservedVariables | ForEach-Object {('$' + $_)}
    $reservedAliases = @(Get-Alias * | Select-Object -Expand Name)

    $variablesRegex = '$\w+(?!.*:)'
    $globalRegex = '(?i)\$bglobal\b:\w+'
    $functionsRegex = '(?i)\bfunction\b[ ]\w+|- \w+|\bfunction\b[ ]\w+'

```

```
$globalScriptVariables = Select-String -Pattern $globalRegex -Input $poshSource -AllMatches |
foreach {$_.Matches} | Select-Object -Unique
```

```
$userDefinedVariables = Select-String -Pattern $variablesRegex -Input $poshSource
-AllMatches | foreach {$_.Matches} | Where-Object {$_ -notin $reservedVariables}
$userDefinedVariables = $userDefinedVariables | Select-Object -Unique
```

```
$userDefinedFunctions = Select-String -Pattern $functionsRegex -Input $poshSource
-AllMatches | foreach {$_.Matches} | Where-Object {$_ -notin $reservedVariables}
$userDefinedFunctions = $userDefinedFunctions | ForEach-Object {$_ -ireplace('function ', '')} |
Select-Object -Unique
```

Clear-Host

```
Write-Host "[GLOBAL VARIABLES]`n"
foreach($globalVariable in $globalScriptVariables) {
    Write-Host $globalVariable
}
Write-Host ('-'*42)
Write-Host "`n"
```

```
Write-Host "[USER VARIABLES]`n"
foreach($userVariable in $userDefinedVariables) {
    Write-Host $userVariable
}
Write-Host ('-'*42)
Write-Host "`n"
```

```
Write-Host "[USER FUNCTIONS]`n"
foreach($userFunction in $userDefinedFunctions) {
    Write-Host $userFunction
}
}
```

```
function Import-PoshSource() {
    Param(
        [Parameter(Position=0, Mandatory=$True)]
        [string] $poshFilePath
    )

    $poshSource = [IO.File]::ReadAllText($poshFilePath)

    try {
        $poshSource = [ScriptBlock]::Create($poshSource)
    }
    catch {}

    return $poshSource
}
```

```
$poshFilePath = 'C:\Users\dev\Desktop\file.ps1'  
$poshSource = Import-PoshSource $poshFilePath
```

```
Get-ScriptVariables $poshSource
```

(Jun 4, 2024 06:41 AM)Ren And Stimpy Wrote: [\[CITE\]](#) Nice share mate!! Always giving back the community!

Very nice contribution for sure!

Happy to see people jumping in on these things.

If anyone has any questions, just LMK!

I might do something similar in a different language soon. Not sure yet.